

3 D O M 2 R E L E A S E ♦ V E R S I O N 2 . 0

Volume 1

- ♦ *3DO M2 Release 2.0 Global Table of Contents*
- ♦ *3DO M2 Release 2.0 Global Index*
- ♦ *3DO M2 Release 2.0 Release Notes*
- ♦ *Getting Started With 3DO M2 Release 2.0*
- ♦ *3DO M2 Debugger Programmer's Guide*
- ♦ *3DO M2 DataStreamer Programmer's Guide*
- ♦ *3DO M2 DataStreamer Reference*
- ♦ *3DO M2 Command List Toolkit*
- ♦ *3DO M2 Video Processing Guide*
- ♦ *3DO M2 FontBuilder User's Guide*
- ♦ *3DO M2 CD-ROM Mastering Guide*



3DO M2 2.0

Global Table of Contents

Version 2.0 – May 1996

Copyright © 1996 The 3DO Company and its licensors.

All rights reserved. This material constitutes confidential and proprietary information of The 3DO Company. This documentation is subject to a license agreement with The 3DO Company and may be used only by parties to such agreement. Use by any other persons, and/or for any purpose not expressly authorized by the agreement, is strictly prohibited.

3DO's LICENSOR(S) MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. 3DO'S LICENSOR(S) DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME JURISDICTIONS. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

Other brand or product names are the trademarks or registered trademarks of their respective holders.

3DO M2 2.0 Global Table of Contents

Volume 1:

3DO M2 Release 1.2 Global Index

3DO M2 Release Notes 1.2

Getting Started With 3DO M2 Release 2.0

Preface

About This Document.....	GSG-v
How This Document Is Organized	GSG-v
Typographical Conventions	GSG-vi

1

3DO M2 Development System

Overview.....	GSG-1
3DO Development Environment.....	GSG-1
.....	GSG-2
3DO M2 Developer's Console System	GSG-2
3DO M2 Developer Documentation Set 2.0	GSG-3
3DO M2 Portfolio and Toolkit Release Version 2.0 CD-ROM	GSG-5
Macintosh Environment.....	GSG-5
Hardware	GSG-5
Software	GSG-5
Optional Additions to the Development Environment	GSG-6
Graphics Development	GSG-6
Audio Development	GSG-6

2

3DO M2 Development Card Installation

About the 3DO M2 Development Card	GSG-8
Installing the 3DO M2 Development Card	GSG-8
Cabling the 3DO M2 Development Card	GSG-10
Installing the M2 2.0 Software	GSG-11

3

Installing the Software

Before you Install the Software	GSG-14
Finding the Online HTML Documentation Files	GSG-14
Using 411 Help files	GSG-15
Installing MPW	GSG-15
Installing the 411 Help files	GSG-15
Installing 3DO M2 2.0 Software	GSG-16
Locating the 3DO Software	GSG-19
Troubleshooting Your Installation	GSG-20
Problems Installing System Extensions	GSG-20

4

Building a 3DO M2 Application

An Overview of the Build Process	GSG-21
The Makefile	GSG-23
A Sample Program	GSG-23
Creating a Makefile with CreateM2Make	GSG-24
Enabling the Debugger in the MakeFile	GSG-25
Building the Program	GSG-25
Launching and Configuring 3DODebug	GSG-26
Running the newview Program	GSG-28
Summary	GSG-29

3DO M2 Debugger Programmer's Guide

Preface

About the 3DO Debugger	DBG-vii
System Requirements	DBG-viii
Contents of This Book	DBG-viii
Typographical Conventions	DBG-viii

1

Introducing the 3DO M2 Debugger

About the 3DO M2 Debugger	DBG-2
Setting Your System Up for Debugging	DBG-2
Configuring Your Source Directories	DBG-4
Configuring Your Application's Makefile	DBG-5
Setting Debugger Preferences	DBG-6
Special Mode	DBG-6

Using the Target Setup Dialog Box	DBG-7
Starting the M2 Debugger	DBG-8
Using the Portfolio Terminal Window	DBG-9
Setting Up Your Source and Symbolic-Information Files	DBG-10
Specifying File Locations with the Directories!Setup Dialog Box	DBG-11
Specifying the Location of Your Application's Source Files	DBG-12
Specifying the Location of Your Application's Symbolic-Information File	DBG-13
Starting a Debugging Session	DBG-13
Setting Up the Debugger for the newview Program	DBG-14
Two Ways to Supply the Location of a File	DBG-15
Executing the debug Command	DBG-16
Using the Source Window	DBG-18
The Disassembly Window	DBG-19
Specifying Source Directories with an .spt File	DBG-22
What's an .spt File?	DBG-22
Summary	DBG-23

2 Using the Debugger

Resuming a Debugging Session	DBG-26
Using BreakPoints	DBG-27
Using the BreakPoints Window	DBG-29
Stepping Through Code	DBG-29
Using the Step In Command	DBG-30
Using the Step Over Command	DBG-31
Working with Variables	DBG-32
Features of the Variables Window	DBG-33
Working with the Variables Window	DBG-33
Two Ways to Use the Variables Window	DBG-34
Examining the Value of a Variable, Method 1	DBG-34
Examining the Value of a Variable, Method 2	DBG-35
Changing the Value of a Variable	DBG-36
Working with an Array	DBG-38
Examining Data	DBG-40
Using the Data Window	DBG-40
Using the Memory Dump Window	DBG-42
Using the Task Window	DBG-43
Viewing and Modifying the Power PC Registers	DBG-43
The Power PC User Registers Window	DBG-44
The Power PC FP Registers Window	DBG-45
The Power PC Supervisor Registers Window	DBG-46
The Stack Crawl Window	DBG-47
The Triangle Engine Disassembly Window	DBG-48
Summary	DBG-50

A

Menu Reference

The File Menu	DBG-52
New	DBG-52
Open (Command + O)	DBG-52
Close (Command +W)	DBG-52
Save (Command + S)	DBG-52
Directories	DBG-53
Page Setup	DBG-53
Print (Command + P)	DBG-53
.Special Mode (Command + =)	DBG-53
Quit (Command + Q)	DBG-54
The Edit Menu.....	DBG-54
Undo	DBG-54
Cut (Command + X)	DBG-54
Copy (Command + V)	DBG-54
Paste	DBG-55
Clear	DBG-55
Select All (Command + A)	DBG-55
Preferences	DBG-55
The View Menu	DBG-55
Disassembly (Command + J)	DBG-56
Data (Command + B)	DBG-56
Source (Command + K)	DBG-56
Variables (Command + M)	DBG-56
Symbols (Command + Y)	DBG-56
BreakPoints (Command + U)	DBG-56
Terminal (Command + T)	DBG-57
Task	DBG-57
Status	DBG-57
PPC User Registers	DBG-57
PPC FP Registers	DBG-57
PPC Supervisor Registers	DBG-57
Stack Crawl	DBG-57
Triangle Disassembly	DBG-57
Send Variable Data (Command + D)	DBG-58
The Execution Menu	DBG-58
Go (Command + G)	DBG-58
Stop (Command + .)	DBG-58
Kill Stopped Task	DBG-58
Step Over (Command + ,)	DBG-58
Step In (Command + ;)	DBG-59
Ignore DebugBreakPoint	DBG-59
Ignore BreakPoints	DBG-59
Initial BreakPoint in task.	DBG-59

The Target Menu	DBG-59
Setup	DBG-59
HW Reset (Command + H)	DBG-59
Memory Dump	DBG-60
Ignore Printf (Command + /)	DBG-60
The Tools Menu	DBG-60
FileSystem Trace	DBG-60
CD Access Log	DBG-60

Index	DBG-61
--------------------	---------------

3DO M2 DataStreamer Programmer's Guide

Preface

About this Document	DSG-xi
Audience	DSG-xi
How this Document is Organized	DSG-xi
Typographical conventions	DSG-xii

1

3DO DataStreamer Overview

For More Information	DSG-1
Why Use the DataStreamer?	DSG-2
Examples of Using the DataStreamer	DSG-2
Preparing a Streaming Application	DSG-3

2

Preparing and Playing a Simple Stream

For More Information	DSG-5
What are Stream Files?	DSG-6
Stream File Preparation Basics	DSG-6
Stream File Terminology	DSG-7
Analyzing Data and Making Trade-Offs	DSG-7
System Resource Limitations	DSG-8
Trade-Offs	DSG-8
Generating and Converting Data	DSG-10
Generating Data	DSG-10
Converting Data to Run-time Formats	DSG-11
Compressing Data	DSG-12
Creating Chunk Files	DSG-13
Creating a Stream File With the Weaver Tool	DSG-13
Creating a Weaver Script	DSG-14
Using the Weaver Tool	DSG-16
Examining the Output Stream	DSG-16
Playing Back a Stream	DSG-16
Data Types and Subscribers	DSG-17
Playback Example Applications	DSG-17

3**Inside Streaming: Basics**

DataStreamer Threads and Data Flow	DSG-20
DataStreamer Threads	DSG-20
Buffer Overview	DSG-21
Playing a Stream—Overview	DSG-22
Allocating Data Buffers	DSG-22
Example: Reading the Stream Header and Allocating Data Buffers	DSG-23
Deciding on Stream Block Size	DSG-24
Creating a Message Port and Message Items	DSG-24
Example: Setting up a Message Port	DSG-24
The Message-Sending Mechanism	DSG-25
Launching the Required Threads	DSG-26
Initializing and Connecting Data Acquisition and Streamer Threads	DSG-27
Initializing and Connecting Subscriber Threads	DSG-28
Extra Initialization for the Audio Subscriber	DSG-29
Registering for End-of-Stream Notification	DSG-30
Starting the Streaming Process	DSG-31
Using Persistent Data With the DATA Subscriber	DSG-31

4**Inside Streaming: Control**

Time and the DataStreamer	DSG-33
The Stream Clock	DSG-33
Changing Time within a Stream	DSG-34
Starting and Stopping	DSG-34
Jumping to a New Location	DSG-36

5**Stream Data Formats**

For More Information	DSG-39
Some Background on MPEG	DSG-39
MPEG Audio	DSG-40
MPEG Video	DSG-40
MPEG Multiplex	DSG-41
Some Background on the Data Streamer	DSG-41
Markers	DSG-42
Subscriber Chunk Common Header Format	DSG-43
Stream Control Chunks (STRM)	DSG-43
MPEG Video Stream Header Chunk	DSG-44
MPEG Video Data Chunks (3 Chunk Subtypes)	DSG-44
Recommendations	DSG-45
MPEG Audio Stream Header Chunk	DSG-46
MPEG Audio Frame Chunks (3 Current and 9 Potential Subtypes)	DSG-47
HALT Chunk	DSG-48
GOTO chunk	DSG-48
STOP Chunk	DSG-50

6 Debugging and Optimization

Debugging with the DataStreamer Libraries	DSG-51
Minimizing Filler	DSG-52
Guidelines for Minimizing Filler	DSG-52
Using Appropriate Streamblock Size and Number of Buffers	DSG-53
Streamblock Size Restrictions	DSG-53
Dealing with Data Rate Spikes	DSG-53
Using More Buffers to Prevent Data Starvation	DSG-54
Selecting Thread Priorities.....	DSG-54
Recommended Order of Process Priorities	DSG-54
How Bad Priorities Can Stop Your Stream	DSG-56
Using Tracing to Understand Data Flow	DSG-56
How to Use Tracing	DSG-57

7 Creating New Subscribers

Introduction.....	DSG-61
Conceptual Background.....	DSG-62
Buffering	DSG-62
Logical Channels	DSG-62
Control Calls	DSG-62
Subscriber Messages	DSG-63
Implementing Subscriber Functionality	DSG-64
Subscriber Message Opcodes	DSG-65
Queued Data and Branching Operations	DSG-65
Video Data	DSG-66
Audio Data	DSG-66

3DO M2 DataStreamer Reference

1 DataStreaming Link Library Calls

Clock

DSClockLE.....	Returns TRUE if (branchNumber1, streamTime1)	
	<= (branchNumber2, streamTime2).	DPR-3
DSClockLT	Returns TRUE if (branchNumber1, streamTime1)	
	< (branchNumber2, streamTime2).	DPR-4
DSGetPresentationClock.....	Returns the Data Stream presentation clock and associated values.	DPR-5
DSSetPresentationClock.....	Sets the Data Stream presentation clock and branch number.	DPR-6

DataSubscriber

GetDataChunk.....	Poll for new data on the specified channel.....	DPR-8
SetDataMemoryFcns.....	Set memory allocation/free functions for a DataSubscriber.....	DPR-9

DSBlockFile

--DSBlockFile-Overview--	Overview of the DSBlockFile module.	DPR-10
AsynchReadBlockFile	Issues an asynchronous read operation on an open BlockFile.	DPR-11
CloseBlockFile	Closes a BlockFile.	DPR-13
CreateBlockFileIOReq	Creates an I/O request Item for a BlockFile.	DPR-14
GetBlockFileBlockSize	Returns the device block size, in bytes, of an open BlockFile.	DPR-15
GetBlockFileSize	Returns the size, in bytes, of an open BlockFile.	DPR-16
OpenBlockFile	Opens a BlockFile.	DPR-17
WaitReadDoneBlockFile	Waits until an I/O request has completed.	DPR-18

DSUtils

CreateMsgItem	Creates a message Item.	DPR-19
NewMsgPort	Creates an unnamed message port.	DPR-20
NewThread	Create a new thread and pass it two arguments.	DPR-21
PollForMsg	Retrieves the next message from the specified message port.	DPR-23

MemPool

--MemPool-Overview--	Overview of the MemPool module.	DPR-24
AllocPoolMem	Allocates an entry from a MemPool.	DPR-25
CreateMemPool	Creates a memory pool (a MemPool).	DPR-26
CreateMemPoolWithOptions	Creates a memory pool (a MemPool) using mem-type options.	DPR-27
DeleteMemPool	Deletes a MemPool.	DPR-28
EnumerateMemPoolEntries	Enumerate all entries in a MemPool.	DPR-29
ForEachFreePoolMember	Enumerate all free entries in a MemPool, e.g. for initialization.	DPR-30
ReturnPoolMem	Returns an entry to a MemPool.	DPR-31

MPEG

FMVCloseDevice	Close an MPEG video device handle.	DPR-32
FMVCreateIOReq	Create an MPEG I/O request setup to signal on I/O completion.	DPR-33
FMVDeleteIOReq	Delete an MPEG I/O request.	DPR-34
FMVGetDeviceItem	Return an FMVDeviceHandle's device Item.	DPR-35
FMVGetPTS	Get the PTS data from a completed MPEG video device read request.	DPR-36
FMVOpenVideo	Open a handle to an MPEG video device channel.	DPR-38
FMVReadVideoBuffer	Read one "frame" of decompressed MPEG video data.	DPR-39
FMVSetVideoMode	Set MPEG video device modes.	DPR-40
FMVSetVideoSize	Set the size parameters for an open MPEG video channel.	DPR-41
FMVWriteBuffer	Write a buffer of compressed data to the MPEG video device.	DPR-42

Shutdown

DisposeDataAcq	Shuts down a data acquisition thread.	DPR-44
DisposeDataStream	Shuts down a data stream thread.	DPR-45

Startup

CreateBufferList	Allocates a stream buffer list for use by the streamer.	DPR-46
FillPoolWithMsgItems	Initialize a MemPool of GenericMsg entries.	DPR-48
FindAndLoadStreamHeader	Load a stream file's stream header chunk into memory.	DPR-49
NewDataAcq	Instantiates a new data acquisition thread.	DPR-50

NewDataStream	Instantiates a new data stream thread.	DPR-51
NewDataSubscriber	Instantiates a DATASubscriber.	DPR-53
NewEZFlixSubscriber	Instantiates an EZFlixSubscriber.	DPR-54
NewMPEGAudioSubscriber	Create a new SAudio subscriber thread.	DPR-55
NewMPEGVideoSubscriber.....	Instantiates an MPEGVideoSubscriber.....	DPR-56
NewSAudioSubscriber	Create a new SAudio subscriber thread.	DPR-57

Streaming

DSCConnect.....	Connects/replaces/disconnects a data acquisition client to the stream.....	DPR-58
DSCControl	Sends a control message to a data stream subscriber.DPR-	DPR-60
DSGetChannel.....	Read the status of a subscriber's logical channel.	DPR-62
DSGoMarker	Asks the streamer to branch within the stream.....	DPR-64
DSIsRunning	Returns TRUE if the Data Stream's presentation clock is currently running.	DPR-67
DSPreRollStream.....	Start prefilling empty buffers with data.	DPR-68
DSSetChannel	Sets the status of a subscriber's logical stream channel.	DPR-70
DSStartStream.....	Starts a stream playing.	DPR-72
DSStopStream.....	Stops stream playback.....	DPR-74
DSSubscribe.....	Add/replace/remove a data stream subscriber.....	DPR-76
DSWaitEndOfStream.....	Register for end-of-stream notification.....	DPR-78

Tracing

AddTrace	Add a trace entry to a trace buffer.....	DPR-80
DumpRawTraceBuffer.....	Dump a tabular listing of all entries in a trace buffer.	DPR-82
DumpTraceCompletionStats...	Dump stats on <start, completion> event pairs from a trace buffer.	DPR-83

2

DataStreaming Tools

AIFFaudio

AudioChunkifier	Chunkifies an AIFF or AIFC sampled audio file.	DPR-87
SAudioTool	Prints or modifies header information of chunkified AIFF files.....	DPR-90

DATA_Subscriber

DATAChunkify.....	Chunkifies a data file for the DATA subscriber.....	DPR-92
-------------------	---	--------

EZFlix

EZFlixChunkifier	Chunkifies an uncompressed QuickTime movie.....	DPR-93
QTVideoChunkifier	Chunkifies an EZFlix-compressed QuickTime movie.....	DPR-94

MPEG

MPEGAudioChunkifier	Chunkifies an MPEG-1 Audio bitstream file.	DPR-95
MPEGVideoChunkifier.....	Chunkifies an MPEG-1 Video elementary stream.....	DPR-96

Utility

DumpStream	Writes diagnostic listing of a stream.....	DPR-98
------------------	--	--------

Weaver

Weaver.....Multiplexes chunkified streams into one playable stream..... DPR-99

Weaver_Script_Commands

audioclockchan.....Specifies which audio subscriber channel will drive the stream clock. DPR-101
dataacqdeltapri.....Specifies the relative priority for the Data Acquisition thread..... DPR-102
enableaudiomask.....Specifies which audio channels to pre-enable. DPR-103
file.....Weaves an input stream file into the woven stream..... DPR-104
markertime.....Adds an entry to the marker table to use as a branch destination..... DPR-105
mediablocksize.....Specifies the block size of the media. (Use 2048 for CD-ROM.) DPR-106
numsubsmessages.....Specifies number of subscriber messages for the
Data Streamer to allocate..... DPR-107
preloadinstrument.....Specifies an audio DSP instrument to preload..... DPR-108
streamblocksize.....Specifies the stream block size used by the Weaver. DPR-109
streambuffers.....Specifies number of stream buffers the application should allocate. DPR-110
streamdeltapri.....Specifies the relative priority for the Data Stream thread..... DPR-111
streamstarttime.....Specifies a time offset for chunks going into the output..... DPR-112
stream. subscriber.....Asks to instantiate a subscribe for a particular data type. DPR-113
writetotochunk.....Writes a STRM GOTO chunk at the specified time in the output stream. .. DPR-114
writemarkertable.....Writes the marker table to the woven output stream. DPR-116
writestopchunk.....Writes a STRM STOP chunk at the specified time in the output stream.... DPR-117
writestreamheader.....Writes the stream header to the woven output stream. DPR-118

3DO M2 Command List Toolkit

1

The Command List Toolkit

Introduction..... CLT-2
Vertex Instructions..... CLT-5
State Control Instructions..... CLT-7
Flow Control Instructions..... CLT-8
Command List Snippets..... CLT-10

2

Graphics State (GState)

Introduction..... CLT-12
Creating and Initializing a GState..... CLT-12
Using a GState..... CLT-13
Synchronizing the Triangle Engine to Video Signals through GState..... CLT-14
Cleanup Routines..... CLT-15
Combining CLT, the 2D and 3D Pipelines and Frameworks,
and the Font Folio..... CLT-15

3 Textures

Introduction	CLT-18
Texture Mapping.....	CLT-19
Using Filtering in Texture-Mapping Operations	CLT-19
Replicating Textures	CLT-19
Levels of Detail (LODs)	CLT-20
Using Mipmaps	CLT-21
Mipmap Filtering.....	CLT-21
How M2 Mipmap Filtering Works	CLT-22
Using Mipmap Filtering	CLT-24
Nearest (Point) Filtering	CLT-24
Linear Filtering	CLT-24
Bi-linear Filtering	CLT-26
Quasi Tri-linear Filtering	CLT-26
Perspective Correction	CLT-27
Computing the U and V Coordinates for Texture Mapping	CLT-27
Using Perspective Off Mode in 2D Operations	CLT-27
How Textures Are Stored in Memory.....	CLT-28
Texture Formats	CLT-29
How Texels Are Stored in Memory	CLT-29
How Uncompressed Texels are Stored	CLT-30
How Compressed Texels are Stored	CLT-30
Special Settings in the Control Byte	CLT-31
PIP Tables	CLT-32
How M2 Interprets Texel Data	CLT-33
How PIP Tables Work	CLT-33
Texture Application Blending.....	CLT-34
Multiplying Color and Alpha Components	CLT-34
Application Modes	CLT-35
Texture Mapping Step-by-Step.....	CLT-35
Loading Textures	CLT-36
Loading Textures by Using MMDMA	CLT-37
Loading Uncompressed Textures	CLT-37
Compressed Texture Load	CLT-39
Texture Mapper Pipeline Register Definitions.....	CLT-40
Command Registers, Fields, and Macros	CLT-40
Texture Generation Registers	CLT-42
Texture Loader Register Definitions	CLT-51

4 Destination Blender

Pixel Discards.....	CLT-60
Pixel Blending.....	CLT-60
Pixel Scaling	CLT-61
Source Blending	CLT-61

Dithering	CLT-62
Z-buffering	CLT-62
Z-banding	CLT-62
Window Clipping	CLT-63
Destination Blender Register Definitions	CLT-63
Register Memory Map	CLT-77

5

Register Setups

Z-Buffering	CLT-81
Dithering	CLT-82
Setting Up the Destination Blender Source Buffer	CLT-82
Transparencies	CLT-82
Texturing	CLT-83
Perspective	CLT-84
Sprites	CLT-84
Lighting	CLT-84
Tiling a Texture	CLT-86
Load an Uncompressed Texture	CLT-86
Loading a PIP Table	CLT-86

6

Sample Code

7

Function Calls

3DO M2 Video Processing Guide

Preface

About this document	VID-ix
Audience	VID-ix
How this document is organized	VID-ix
Typographical conventions	VID-x

1

Overview

Introduction	VID-1
Chapter Overview	VID-2
The Main Paths Available	VID-2
What You Start With	VID-2
The Overall Process	VID-2
The Main Paths You can Take - MPEG or EZFlix (and DSP Audio)	VID-3
Factors in Choosing a Path	VID-3
Diagram of Video Processing Paths	VID-5

The MPEG Path	VID-5
Overview	VID-5
Hardware MPEG Encoding With No Editing	VID-6
Hardware MPEG Encoding With Editing	VID-7
Software MPEG Encoding	VID-8
The EZFlix Path	VID-10
The DSP Audio Path	VID-11

2 Digitizing

Introduction.....	VID-13
Source Material	VID-14
Shooting Your Own Video	VID-14
Film-Based Source	VID-14
Digitizing Video.....	VID-15
Digitizing Process	VID-16
Processing Digitized Materials.....	VID-17
3-2 Pulldown	VID-18
Deinterlacing	VID-18
Cropping and Resizing	VID-18

3 Compression

Introduction.....	VID-21
Which Compression Algorithm.....	VID-21
EZFlix	VID-22
Cinepak (Not Currently Supported in 3DO M2)	VID-23
MPEG	VID-23
QuickTime and Cinepak Tools.....	VID-24
Size Constraints	VID-24
EZFlix Advantages	VID-24
Setting Data Rate	VID-25
Compression Options.....	VID-25

4 MPEG Encoding with Sparkle

Introduction.....	VID-27
Overview	VID-27
Related Documentation	VID-28
About Sparkle	VID-28
Requirements	VID-28
Preprocessing	VID-28
Some Considerations	VID-29

Encoding with Sparkle	VID-30
Setting MPEG Playback Preferences	VID-30
Setting MPEG Encoding Parameters	VID-32
Fine Tuning	VID-33
Putting MPEG into a Stream	VID-35

5

Video Tools

Introduction	VID-37
The 3-2 Pulldown Tool	VID-37
Understanding Telecine	VID-38
Choosing a Movie	VID-39
Field Dominance	VID-39
Removing Composite Frames	VID-40
Saving a Processed Movie	VID-42
Troubleshooting	VID-42
MovieEdit	VID-43
Saving Movies with MovieEdit	VID-43
MovieEdit User Interface	VID-44
Limitations	VID-45
Tips and Tricks	VID-45
MovieCompress	VID-45
Movie Compression Settings Dialog Options	VID-45
Using the MovieCompress Tool	VID-46
EZFlixChunkifier	VID-48
EZFlixChunkifier Command-Line Arguments	VID-48
Weaving	VID-49

6

Compressing with EZFlix

Introduction	VID-51
About this Chapter	VID-51
Audience	VID-52
EZFlix Functionality	VID-52
Caveats	VID-53
Installation	VID-53
Compressing a Movie	VID-53
The Weave Script	VID-53
Playing a Movie	VID-54
Player Controls	VID-55

Index	VID-57
-------------	--------

3DO M2 FontBuilder User's Guide

Preface

About This Document	FBR-vii
Audience	FBR-vii
How This Document is Organized	FBR-vii
Related Documentation	FBR-vii
Typographical Conventions	FBR-vii

1

Using M2 FontBuilder

Introduction	FBR-1
Chapter Overview	FBR-1
Creating an Anti-Aliased M2 FontBuilder Font	FBR-1
Saving Fonts	FBR-4
Using the "Other" Menu	FBR-5

3DO M2 CD-ROM Mastering Guide

Preface

Audience	CDM-xi
How This Document is Organized	CDM-xi
How to Work With This Document	CDM-xii
Typographical Conventions	CDM-xii

1

Hardware and Software Requirements

Requirements for a 3DO M2 Development System	CDM-2
3DO Development System Hardware Requirements	CDM-2
3DO Development System Software Requirements	CDM-3
Requirements for a CD-ROM Recording Station	CDM-3
Recording Station Layout	CDM-3
Hardware Requirements	CDM-4
Software and Media	CDM-4

2

Preparing for the Mastering Process

Application Size Limitations	CDM-6
Factors Influencing Available Space	CDM-6
Determining Available Space for Your Application	CDM-6
Programming Notes	CDM-7
Using Relative Pathnames	CDM-7
Hints on Error Checking and Other Advice	CDM-7

3**Creating and Testing the Image File**

Background and Overview	CDM-10
Layout Tool Background Information	CDM-10
Optimized Layout Process Overview	CDM-12
Required Files and Folders.....	CDM-12
Preparing Your Title's Files for the Layout Process	CDM-12
Modifying the BannerScreen File	CDM-13
Preparing Simple cdrom.image Files	CDM-13
Editing the cdrom.tcl File for the Simple Image	CDM-14
Creating the Simple Image	CDM-15
Testing the Simple Image	CDM-15
Preparing an Optimized CD-ROM Image.....	CDM-16
Creating the Base Image for Information Collection	CDM-16
Preparing the Debugger for Information Collection	CDM-16
Running Your Title to Collect Access Information	CDM-17
Running the Layout Tool to Create an Optimized Image	CDM-17
Testing the CD-ROM Image File on a Macintosh	CDM-20
Layout Tool Troubleshooting.....	CDM-20

4**Mastering and Testing the CD-ROM**

Preparation and Setup for the Mastering Process	CDM-24
Maximizing Recording Speed	CDM-24
Setting Up the QuickTOPIX Software	CDM-24
Creating a CD-ROM Disc for Testing	CDM-27
Testing the CD-ROM Disc Using the Debugger.....	CDM-28
Debugging Hints	CDM-28
Creating a Master CD-ROM Disc.....	CDM-29

5**Delivery for Authorization**

About the Authorization Requirements	CDM-32
--	--------

A**CD-ROM Mastering Checklist**

Reading List.....	CDM-33
Preparing for the Mastering Process	CDM-33
Creating and Testing a Simple CD-ROM Image File.....	CDM-34
Creating an Optimized Image.....	CDM-34
Testing the Optimized CD-ROM Image on the Macintosh	CDM-36
Mastering the CD-ROM Disc.....	CDM-36
Testing the CD-ROM.....	CDM-38
Preparing Materials for Authorization	CDM-38

B

CD-ROM Mastering Etiquette

CD-ROM Basics	CDM-40
Constant Linear Velocity	CDM-40
3DO Drive Specifications	CDM-40
Data Rate Issues	CDM-41
Error Detection and Correction	CDM-41
Programming Do's and Don'ts	CDM-42
Making Assumptions about Data Delivery Rates	CDM-42
Handling Data Delays	CDM-42
Alternating File Reads	CDM-42
Testing Performance	CDM-43
Other Things You Should Do	CDM-43
Head Seeks and CD-ROM Mechanism Lifetime	CDM-44
Optimization Issues	CDM-44

Volume 2:**3DO M2 Graphics Tools****Preface**

About This Book	MGT-xi
About the Audience.....	MGT-xi
How This Book Is Organized.....	MGT-xi
Related Documentation	MGT-xii
Typographical Conventions	MGT-xii

1**M2 Graphics Tools**

Overview.....	MGT-2
3D Art Conversion Tools.....	MGT-2
3D Studio to SDF Converter	MGT-2
Alias to SDF Converter	MGT-2
Strata StudioPro to SDF Converter	MGT-3
Lightwave to SDF Converters	MGT-3
Geometry Compiler.....	MGT-3
Texture Tools and Texture Library	MGT-3

2**Displaying 3D Models on M2**

File Formats	MGT-5
The File Conversion Process.....	MGT-6
Creating an ASCII SDF File	MGT-6
Keyframe Animation Conversion	MGT-7
Optimizing Triangle Drawing Performance	MGT-8
Limit Characters	MGT-8
Flatten The Hierarchy	MGT-8
Share Vertex Data	MGT-8
Converting Textures To UTF	MGT-9
Slicing and Dicing with the Geometry Compiler.....	MGT-10
Viewing the Model	MGT-11
Viewing the Animation.....	MGT-12

3**3D Art Conversion Tools**

Introduction.....	MGT-14
SDF Files	MGT-14
Keyframe Animation Data	MGT-14
3D Studio to SDF Converter (3dstosdf).....	MGT-15
Additional Notes	MGT-16
Alias to SDF (altosdf).....	MGT-17
Additional Notes	MGT-18

Strata StudioPro to SDF Converter (ssptosdf)	MGT-19
Additional Notes	MGT-21
Lightwave to SDF Converters (lwtosdf & lwstoanim).....	MGT-22
Lightwave to SDF (lwstosdf) Conversion Behavior	MGT-22
Lightwave Scene to Animation (lwstoanim) Conversion Behavior	MGT-23

4

Geometry Compiler

Overview	MGT-26
Compiling Textures.....	MGT-26
Slicing and Dicing	MGT-26
Wrapping Texture Coordinates	MGT-27
Binary SDF Writer	MGT-28
Portable Binary SDF Format	MGT-28
Non-Portable Binary SDF Format	MGT-28
Pre-Lit Geometry Writer	MGT-28
Snake Primitives	MGT-29

5

M2 Texture Tools and Library

Overview	MGT-32
M2 Graphics Tools	MGT-32
Texture Tool Library	MGT-32

A

Conversion Tools and Geometry Compiler

3dstosdf	MGT-36
altosdf	MGT-37
ssptosdf	MGT-38
lwtosdf	MGT-39
lwstoanim	MGT-40
gcomp	MGT-41

B

Texture Tool Calls

ppmtoutf	MGT-45
psdtoutf	MGT-46
quantizer	MGT-47
quanttopip	MGT-48
sgitoutf	MGT-49
utfaddlod	MGT-50
utfcompress	MGT-51
utffit	MGT-53
utfflip	MGT-55
utfinfo	MGT-56
utflitdown	MGT-57
utfmakelod	MGT-58

utfmakepip	MGT-59
utfmakesame	MGT-60
utfmerge	MGT-61
utfmipcat	MGT-62
utfmodpip	MGT-64
utfpopfine	MGT-65
utfresize	MGT-66
utfstrip	MGT-67
utftoppm	MGT-68
utfuncompress	MGT-69
utfunmip	MGT-70

C

Texture Library Calls

M2TXColor_Create	MGT-78
M2TXColor_Decode	MGT-79
M2TXColor_GetAlpha	MGT-80
M2TXColor_GetBlue	MGT-81
M2TXColor_GetGreen	MGT-82
M2TXColor_GetRed	MGT-83
M2TXColor_GetSSB	MGT-84
M2TXDCI_GetADepth	MGT-85
M2TXDCI_GetCDepth	MGT-86
M2TXDCI_GetColorConst	MGT-87
M2TXDCI_GetFHasAlpha	MGT-88
M2TXDCI_GetFHasColor	MGT-89
M2TXDCI_GetFHasSSB	MGT-90
M2TXDCI_GetFIsLiteral	MGT-91
M2TXDCI_GetFIsTrans	MGT-92
M2TXDCI_GetSize	MGT-93
M2TXDCI_GetTexFormat	MGT-94
M2TXDCI_SetADepth	MGT-95
M2TXDCI_SetCDepth	MGT-96
M2TXDCI_SetColorConst	MGT-97
M2TXDCI_SetFHasAlpha	MGT-98
M2TXDCI_SetFHasColor	MGT-99
M2TXDCI_SetFHasSSB	MGT-100
M2TXDCI_SetFIsLiteral	MGT-101
M2TXDCI_SetFIsTrans	MGT-102
M2TXDCI_SetTexFormat	MGT-103
M2TXFilter_GetWidth	MGT-104
M2TXFilter_Reset	MGT-105
M2TXFilter_SetWidth	MGT-106
M2TXFormat_GetADepth	MGT-107
M2TXFormat_GetCDepth	MGT-108
M2TXFormat_GetFHasAlpha	MGT-109

M2TXFormat_GetFHasColor	MGT-110
M2TXFormat_GetFHasSSB	MGT-111
M2TXFormat_GetFlsLiteral	MGT-112
M2TXFormat_GetFlsTrans	MGT-113
M2TXFormat_SetADepth	MGT-114
M2TXFormat_SetCDepth	MGT-115
M2TXFormat_SetFHasAlpha	MGT-116
M2TXFormat_SetFHasColor	MGT-117
M2TXFormat_SetFHasSSB	MGT-118
M2TXFormat_SetFlsLiteral	MGT-119
M2TXFormat_SetFlsTrans	MGT-120
M2TXHeader_FreeLODPtr	MGT-121
M2TXHeader_FreeLODPtrs	MGT-122
M2TXHeader_GetADepth	MGT-123
M2TXHeader_GetBorder	MGT-124
M2TXHeader_GetCDepth	MGT-125
M2TXHeader_GetColorConst	MGT-126
M2TXHeader_GetFHasAlpha	MGT-127
M2TXHeader_GetFHasColor	MGT-128
M2TXHeader_GetFHasColorConst	MGT-129
M2TXHeader_GetFHasNoDAB	MGT-130
M2TXHeader_GetFHasNoDCI	MGT-131
M2TXHeader_GetFHasNoLR	MGT-132
M2TXHeader_GetFHasNoTAB	MGT-133
M2TXHeader_GetFHasPIP	MGT-134
M2TXHeader_GetFHasSSB	MGT-135
M2TXHeader_GetFlsCompressed	MGT-136
M2TXHeader_GetFlsLiteral	MGT-137
M2TXHeader_GetFlags	MGT-138
M2TXHeader_GetLODDim	MGT-139
M2TXHeader_GetLODLength	MGT-140
M2TXHeader_GetLODPtr	MGT-141
M2TXHeader_GetMinXSize	MGT-142
M2TXHeader_GetMinYSize	MGT-143
M2TXHeader_GetNumLOD	MGT-144
M2TXHeader_GetSize	MGT-145
M2TXHeader_GetTexFormat	MGT-146
M2TXHeader_GetVersion	MGT-147
M2TXHeader_GetVersion	MGT-148
M2TXHeader_SetBorder	MGT-149
M2TXHeader_SetCDepth	MGT-150
M2TXHeader_SetColorConst	MGT-151
M2TXHeader_SetFHasAlpha	MGT-152
M2TXHeader_SetFHasColor	MGT-153
M2TXHeader_SetFHasColorConst	MGT-154
M2TXHeader_SetFHasNoDAB	MGT-155

M2TXHeader_SetFHasNoDCI	MGT-156
M2TXHeader_SetFHasNoLR	MGT-157
M2TXHeader_SetFHasNoTAB	MGT-158
M2TXHeader_SetFHasPIP	MGT-159
M2TXHeader_SetFHasSSB	MGT-160
M2TXHeader_SetFIsCompressed	MGT-161
M2TXHeader_SetFIsLiteral	MGT-162
M2TXHeader_SetFlags	MGT-163
M2TXHeader_SetLODPtr	MGT-164
M2TXHeader_SetMinXSize	MGT-165
M2TXHeader_SetMinYSize	MGT-166
M2TXHeader_SetNumLOD	MGT-167
M2TXHeader_SetTexFormat	MGT-168
M2TXHeader_SetVersion	MGT-169
M2TXIndex_Alloc	MGT-170
M2TXIndex_Compress	MGT-171
M2TXIndex_FindBestDCI	MGT-172
M2TXIndex_FindBestDCIPIP	MGT-173
M2TXIndex_Free	MGT-174
M2TXIndex_Init	MGT-175
M2TXIndex_LODCreate	MGT-176
M2TXIndex_ReorderToPIP	MGT-177
M2TXIndex_SmartFormatAssign	MGT-178
M2TXIndex_ToUncompr	MGT-179
M2TXPIP_GetColor	MGT-180
M2TXPIP_GetIndexOffset	MGT-181
M2TXPIP_GetNumColors	MGT-182
M2TXPIP_GetSize	MGT-183
M2TXPIP_GetSortIndex	MGT-184
M2TXPIP_MakeTranslation	MGT-185
M2TXPIP_SetColor	MGT-186
M2TXPIP_SetIndexOffset	MGT-187
M2TXPIP_SetNumColors	MGT-188
M2TXPIP_SetSortIndex	MGT-189
M2TXRaw_Alloc	MGT-190
M2TXRaw_Compress	MGT-191
M2TXRaw_DitherDown	MGT-192
M2TXRaw_FindPIP	MGT-193
M2TXRaw_Free	MGT-194
M2TXRaw_GetColor	MGT-195
M2TXRaw_Init	MGT-196
M2TXRaw_LODCreate	MGT-197
M2TXRaw_MakePIP	MGT-198
M2TXRaw_MapToPIP	MGT-199
M2TXRaw_ToUncompr	MGT-200
M2TX_Compress	MGT-201

M2TX_ComprToM2TXIndex	MGT-202
M2TX_ComprToM2TXRaw	MGT-203
M2TX_Extract	MGT-204
M2TX_Free	MGT-205
M2TX_GetDCI	MGT-206
M2TX_GetHeader	MGT-207
M2TX_GetPIP	MGT-208
M2TX_Init	MGT-209
M2TX_IsLegal	MGT-210
M2TX_Print	MGT-211
M2TX_ReadFile	MGT-212
M2TX_ReadFileNoLODs	MGT-213
M2TX_ReadMacFile	MGT-214
M2TX_SetDCI	MGT-215
M2TX_SetDefaultXWrap	MGT-216
M2TX_SetDefaultYWrap	MGT-217
M2TX_SetHeader	MGT-218
M2TX_SetPIP	MGT-219
M2TX_UncomprToM2TXIndex	MGT-220
M2TX_UncomprToM2TXRaw	MGT-221
M2TX_WriteFile	MGT-222
M2TX_WriteMacFile	MGT-223

D

3D0 M2 Texture File Format

Introduction.....	MGT-226
File Format.....	MGT-226
Form Type	MGT-227
Header Chunk	MGT-227
Pen Index Palette Chunk	MGT-232
Data Compression Information Chunk	MGT-233
Texel Data Chunk	MGT-235
Texture Attributes Chunk	MGT-240
Texture Load Rects Chunk	MGT-241
Destination Blend Attributes Chunk	MGT-243
Texture Page Chunk	MGT-244
Page CLT Chunk	MGT-249
Multiple Texture Files.....	MGT-251
Animation	MGT-254
Form Type	MGT-254
Frame Container Chunk	MGT-254
Sequences Container Chunk	MGT-254
Sequence Action Chunk	MGT-254
Sequence Play Chunk	MGT-255

3DO PostPro 2.0 User's Guide

Preface

About This Document.....	PRO-vii
About the Audience.....	PRO-vii
How This Document is Organized.....	PRO-vii
Related Documents	PRO-viii
Typographical Conventions	PRO-viii

1

Before You Begin

Overview.....	PRO-1
About PostPro 2.0	PRO-2
Feature Overview	PRO-2
AppleScript	PRO-2
Drag and Drop	PRO-2
Supported file types	PRO-2
Hardware and Software Prerequisites.....	PRO-2
System Hardware Specifications	PRO-3
Software Specifications	PRO-3
Installation Issues.....	PRO-3

2

Using the Viewer

Overview.....	PRO-5
About the Image Viewer.....	PRO-5
About the Document Window.....	PRO-6

3

Converting Files

Overview.....	PRO-9
About Converting Files.....	PRO-10
Converting to a Texture	PRO-10
Using AppleScript for Batch Processing.....	PRO-12

4

PostPro 2.0 Reference

Overview.....	PRO-13
Menus.....	PRO-14
Menu Bar	PRO-14
File Menu	PRO-14
Edit Menu	PRO-15
Document Menu	PRO-15
Windows and Dialog Boxes	PRO-16
Image Viewer Window	PRO-16
Photoshop to Texture Dialog Box	PRO-16

3DO M2 Graphics Programmer's Guide

Preface

Animating a Scene.....	GPG-xviii
Frame-by-Frame Animation	GPG-xviii
About This Book	GPG-xviii
This Book and You	GPG-xix
How This Book Is Organized.....	GPG-xix
Related Documentation	GPG-xix
Typographical Conventions	GPG-xx

1

Introducing M2 Graphics

The Graphics Framework.....	GPG-2
The Graphics Pipeline.....	GPG-2
The Graphics Folio.....	GPG-3
The Scene Description Format.....	GPG-3

2

The Graphics Framework

Object Hierarchy	GPG-7
SDF Files.....	GPG-9
How SDF Files Work	GPG-10
SDF Housekeeping Functions	GPG-11
SDF Iterators	GPG-13
Initializing and Displaying Graphics Framework Objects	GPG-15
Characters	GPG-16
Character Attributes	GPG-17
Character Transformation Operations	GPG-18
Rotating Characters	GPG-26
Functions for Transforming Characters with Matrices	GPG-27
Prioritizing Multiple-Character Operations	GPG-27
Bounding Boxes	GPG-30
Models: Characters Described Geometrically	GPG-34
Models	GPG-34
Material Properties	GPG-36
Summary of Material Properties	GPG-38
Putting Characters in a Hierarchy	GPG-38
Character Hierarchy Functions	GPG-42
Character Hierarchy Iterators	GPG-44
Scenes.....	GPG-45
Dynamic Characters and Static Characters	GPG-47
Scene Attributes	GPG-47
Functions for Working with Scenes	GPG-47
Cameras	GPG-50
Camera Attributes	GPG-51
Functions for Working with Cameras	GPG-51

Graphics Framework Lighting Operations	GPG-53
Directional Lighting	GPG-54
Point Lighting	GPG-55
Spotlights	GPG-55
Ambient Lighting	GPG-56
Lighting Attributes	GPG-56
Functions for Working with Lighting	GPG-57
Framework Animation Engines	GPG-58
The Engine Object	GPG-59
Engine Attributes	GPG-59
Using Engines	GPG-60
How aSimple Engine Works	GPG-60
Arrays	GPG-61
Functions	GPG-62
Extensibility	GPG-65
GfxObj: The Common Ancestor	GPG-65
GfxObj Functions	GPG-65
Constructing, Destroying, Copying, and Printing Subclasses	GPG-66
Runtime Type-Checking Functions	GPG-66
Reference Counting	GPG-67
Defining New Graphics Framework Classes	GPG-67
Graphics Utility Functions	GPG-72
Graphics Framework Extensions in SDF Files	GPG-72
Print-Level Routines	GPG-75
Print-Level Options	GPG-75
Print-Level Functions	GPG-75

3

The 2D Graphics Framework

2D Framework Objects	GPG-80
Sprite Objects	GPG-80
Names of 2D Framework Objects	GPG-81
The 2D Framework and the GState Object	GPG-81
How 2D Framework Objects Are Grouped Together for Processing	GPG-82
The Sprite Object	GPG-82
Drawing Sprites	GPG-83
Rotating and Scaling Sprites	GPG-83
Rendering Sprites in Strips and Sections	GPG-85
Attributes of Sprite Objects	GPG-86
ExtSpriteObject Attributes	GPG-88
Sprite-Object Functions	GPG-88
The Grid Object	GPG-93
Functions for Creating, Deleting, and Working with Grid Objects	GPG-93

Rendering 2D Objects.....	GPG-94
Other 2D Primitives: Points, Lines, Triangles, and Rectangles	GPG-95
Special Considerations in Rendering 2D Objects	GPG-96
Example: Creating a Sprite Object.....	GPG-100

4 The Graphics Pipeline

The Graphics Pipeline and the Graphics Framework.....	GPG-106
Graphics Pipeline Objects	GPG-107
How the GP Uses Texture objects and Surface Objects	GPG-109
Other GP Objects	GPG-109
Attributes of GP Objects	GPG-111
Getting an Object's Type	GPG-111
Functions for Handling GP Attributes	GPG-111
Primitive Drawing Attributes	GPG-113
Functions for Handling Rendering Attributes	GPG-113
Using the GP's Rendering Primitives	GPG-114
Rendering-Control Attributes	GPG-117
Rendering-Control Functions	GPG-117
Using 3D Points and Vectors.....	GPG-118
Performing Transformations on Points	GPG-119
Performing Transformations on Vectors	GPG-120
Macros for Performing Operations on Points and Vectors	GPG-121
Lighting	GPG-123
Architecture of the LightProp Structure	GPG-123
LightProp Functions	GPG-124
Combining Lighting and Colors	GPG-124
Computing Colors Under Various Lighting Conditions	GPG-125
Two-sided Lighting	GPG-128
Material (MatProp) Functions	GPG-128
Transformations	GPG-129
Functions for Performing Transformation Operations	GPG-129
Functions for Performing Geometry Pipeline Transformations	GPG-135
The GP's Geometry Structure	GPG-136
The QuadMesh Structure	GPG-139
Surfaces	GPG-141
Rendering of Previously Constructed Surfaces	GPG-142
Functions for Rendering Surfaces	GPG-143
Textures.....	GPG-144
The Texture API	GPG-144
Texture Mapping	GPG-145
Mipmapping	GPG-146
Textures and the M2 Hardware	GPG-147
How Textures Are Stored in Memory	GPG-148
Texture Formats	GPG-148
Textures and Surfaces	GPG-150

The Texture object	GPG-150
PIP Tables	GPG-151
Mipmap Filtering	GPG-154
Texture Application Blending	GPG-158
Texture Mapping Step by Step	GPG-159
The TexBlend Object	GPG-159
The Load Rectangle	GPG-162
Functions for Performing Texture-Related Operations	GPG-162
Destination Blending.....	GPG-173
How Destination Blending Works	GPG-173
Pixel Scaling	GPG-175
Source Image Specification	GPG-175
Z-Buffer Functions	GPG-175
Dithering Functions	GPG-177
Window Clipping	GPG-178
Setting and Getting Destination-Blend Attributes	GPG-181
Specifying the 'A' Path for the Destination Blender	GPG-183
Controlling How the Channels Are Blended	GPG-190
Alpha Clamping	GPG-190
Blending with a Source Frame Buffer	GPG-193

5

The Graphics Folio

The Graphics Folio API	GPG-195
About the Graphics Folio	GPG-197
Graphics Folio Items.....	GPG-197
Bitmaps	GPG-198
The Bitmap Item	GPG-198
Bitmap Item Fields	GPG-199
Other Bitmap Properties	GPG-200
Adjusting Bitmap Parameters	GPG-201
Allocating Buffer Memory	GPG-202
Views.....	GPG-204
Display Coordinates	GPG-204
The View Item	GPG-205
Width and PixWidth	GPG-207
Defaults	GPG-207
Creating a View	GPG-208
Showing a Subregion of a Bitmap	GPG-208
The ViewTypeInfo Structure	GPG-209
ViewLists	GPG-210
TEContexts	GPG-210
The TEContext item	GPG-211
Projectors.....	GPG-211
The Projector Item	GPG-212
Item Modification.....	GPG-213

Display Architecture	GPG-214
View and ViewList Hierarchy	GPG-215
Creating Views	GPG-215
Making a View Visible	GPG-216
View Signals	GPG-217
Using View Signals	GPG-217
Blocking Signals	GPG-223
Display Construction and Compilation.....	GPG-223
Optimizing Compilations	GPG-224
The Default Display and Other Displays	GPG-224
Graphics Folio Tips and Tricks.....	GPG-224
Managing Your Views	GPG-224
Display Locking	GPG-225
Double-Buffering	GPG-225

6 SDF Files

Introduction.....	GPG-228
How SDF Files Work	GPG-229
Binary and ASCII SDF Files	GPG-229
Classes of Objects in SDF Files.....	GPG-229
Enumerations and BitFields	GPG-230
Writing SDF Files	GPG-231
Including Other Files in SDF Files	GPG-231
Case Considerations	GPG-231
Punctuation in SDF Files	GPG-231
Data Types in SDF Files	GPG-231
Version Numbers of SDF Files	GPG-233
Units	GPG-233
Shapes Used in SDF Files	GPG-233
Tessellation	GPG-233
Blocks and Cubes	GPG-233
Cylinders and Cones	GPG-234
Spheres and Ellipses	GPG-237
Torus	GPG-237
Vertex Primitives	GPG-238
Vertex Order	GPG-241
Materials in SDF Files	GPG-242
Textures in SDF Files	GPG-242
Texture Generation	GPG-242
The TexBlend Class	GPG-244
Surfaces in SDF Files	GPG-245
Defining a Surface	GPG-246
Arrays	GPG-247
Faceted Primitives	GPG-247

SDF Scenes	GPG-248
Characters	GPG-248
Cameras	GPG-249
Lights	GPG-249
Models	GPG-249
Examples: Using SDF Files	GPG-249

A

M2 Graphics Hardware

M2 Hardware Components	GPG-256
The Integrated System.....	GPG-257
Video Standards	GPG-257
NTSC Display Basics	GPG-257
PAL Display Basics	GPG-260

B

SDF Class Descriptions

3DO M2 Graphics Programmer's Reference

1

2D Framework Link Library Calls

CltPipControlBlock_Create.....Create a PIP controlblock.	GPR-3
CltPipControlBlock_Delete.....Delete a PIP control block.....	GPR-4
CltTxData_Create.....Create a CltTxData structure that describes texel data in memory	GPR-5
CltTxData_Delete.....Delete a CltTxData structure	GPR-6
F2_ColoredLines.....Draw colored lines on the display	GPR-7
F2_CopyRect.....Copy a rectangle from memory to the display	GPR-8
F2_Draw.....Draw a 2D object on the display	GPR-9
F2_DrawLine.....Draw a shaded line on the display	GPR-10
F2_DrawList.....Draw a 2D object on the display	GPR-11
F2_FillRect.....Draw a filled rectangle on the display	GPR-12
F2_Point.....Draw a single pixel.....	GPR-13
F2_Points.....Draw colored pixels	GPR-14
F2_ShadedLines.....Draw shaded lines on the display	GPR-15
F2_Triangles.....Draw 2D triangles	GPR-16
F2_TriFan.....Draw a 2D triangle fan	GPR-17
F2_TriStrip.....Draw a 2D triangle strip	GPR-18
Gro_Create.....Create a grid object.....	GPR-19
Gro_Delete.....Delete a grid object.....	GPR-21
Gro_GetHDelta.....Obtain the hdelta of the grid object.....	GPR-22
Gro_GetHeight.....Query the height of the sprite array in a grid object.....	GPR-23
Gro_GetPosition.....Obtain the screen position of the grid object.....	GPR-24
Gro_GetSpriteArray.....Get a pointer to the sprite array associated with a grid object	GPR-25
Gro_GetVDelta.....Obtain the vdelta of the grid object.....	GPR-26
Gro_GetWidth.....Query the width of the sprite array in a grid object	GPR-27
Gro_SetHDelta.....Set the hdelta of the grid object.....	GPR-28

Gro_SetHeight	Set the height of the sprite array associated with a grid object.....	GPR-29
Gro_SetPosition	Set the screen position of the grid object	GPR-30
Gro_SetSpriteArray.....	Attach a sprite array to a GridObj.....	GPR-31
Gro_SetVDelta	Set the vdelta of the grid object	GPR-32
Gro_SetWidth	Set the width of the sprite array associated with a grid object.....	GPR-33
Spr_Create.....	Create a sprite object.....	GPR-34
Spr_CreateExtended	Create a an extended sprite object.....	GPR-35
Spr_CreatePipControlBlock	Create a PIP control block and attach it to a sprite.....	GPR-36
Spr_CreateShort	Create a short sprite object.....	GPR-37
Spr_CreateTxData.....	Create a CltTxData block and attach it to a sprite.....	GPR-38
Spr_Delete	Delete a sprite object.....	GPR-39
Spr_DisableClipping	Disables clipping for a sprite.....	GPR-40
Spr_EnableClipping	Enables clipping for a sprite.....	GPR-41
Spr_GetColors	Get the colors at the 4 corners of an extended sprite object.....	GPR-42
Spr_GetCorners.....	Query the locations of the corners of a sprite object.....	GPR-43
Spr_GetDBlendAttr	Get the setting of a destination blend attribute in the sprite object	GPR-44
Spr_GetGeometryEnable.....	Query the geometry enable state of the sprite object.....	GPR-45
Spr_GetHeight	Query the default height of the sprite object.....	GPR-46
Spr_GetHSlice	Query the hslice value of the sprite object.....	GPR-47
Spr_GetPIP	Get the address of the PIP table associated with a sprite object.....	GPR-48
Spr_GetPosition	Obtain the position of the sprite object.....	GPR-49
Spr_GetSkip	Query the skip flag of the sprite object.....	GPR-50
Spr_GetTextureAttr	Get the setting of a texture attribute in the sprite object	GPR-51
Spr_GetTextureData	Attach texture data to a sprite object.....	GPR-52
Spr_GetVSlice.....	Query the vslice value of the sprite object.....	GPR-53
Spr_GetWidth	Query the default width of the sprite object.....	GPR-54
Spr_GetZValues.....	Get the Z values at the 4 corners of an extended sprite object.....	GPR-55
Spr_LoadTexture	Loads one or more textures from a UTF file.....	GPR-56
Spr_LoadUTF.....	Load a sprite from a UTF file.....	GPR-58
Spr_MapCorners	Set the position and corners of a sprite object.....	GPR-59
Spr_RemoveDBlendAttr.....	Remove a destination blend attribute in the sprite object	GPR-60
Spr_RemovePIP	Remove the PIP from a sprite object	GPR-61
Spr_RemoveTextureAttr	Remove a texture attribute from the sprite object.....	GPR-62
Spr_RemoveTextureData.....	Remove the texture data from a sprite object.....	GPR-63
Spr_ResetCorners	Reset the corners of a sprite object	GPR-64
Spr_Rotate	Rotate the sprite counterclockwise by the specified angle.....	GPR-65
Spr_Scale	Change the size of a sprite object by specified ratios.....	GPR-66
Spr_SetColors	Set the colors at the 4 corners of an extended sprite object.....	GPR-67
Spr_SetCorners.....	Set the locations of the corners of a sprite object.....	GPR-68
Spr_SetDBlendAttr	Set a destination blend attribute in the sprite object	GPR-69
Spr_SetGeometryEnable.....	Enable/disable the geometry on a sprite object.....	GPR-70
Spr_SetHeight	Set the default height of the sprite object.....	GPR-71
Spr_SetHSlice.....	Set the hslice value of the sprite object.....	GPR-72
Spr_SetPIP	Set the PIP table to be loaded for a sprite object.....	GPR-73
Spr_SetPosition.....	Set the position of the sprite object.....	GPR-74
Spr_SetSkip.....	Set the state of the skip flag of the sprite object.....	GPR-75

Spr_SetTextureAttr	Set a texture attribute in the sprite object	GPR-76
Spr_SetTextureData	Attach texture data to a sprite object.	GPR-77
Spr_SetVSlice	Set the vslice value of the sprite object.	GPR-78
Spr_SetWidth	Set the default width of the sprite object.	GPR-79
Spr_SetZValues	Set the Z values at the 4 corners of an extended sprite object.	GPR-80
Spr_Translate	Change the position of a sprite object by specified deltas.	GPR-81
Spr_UnloadTexture	Frees resources associated with previously loaded texture files	GPR-82

2

3D Framework Link Library Calls

frame

Char_Append	Append a Character at the end of this Group.	GPR-85
Char_AxisRotate	Rotate the character in the world coordinate system about an arbitrary axis.	GPR-86
Char_Clone	Make a copy of a character hierarchy	GPR-87
Char_CloneDeep	Make a copy of a character hierarchy and its attributes	GPR-88
Char_Create	Creates a new character	GPR-89
Char_Delete	Deletes a character	GPR-90
Char_Display	Display a character on a GP	GPR-91
Char_First	Returns the first child of a character	GPR-92
Char_ForAll	Initialize a character hierarchy iterator	GPR-93
Char_GetAt	Returns the Nth child	GPR-94
Char_GetBound	Finds a character's bounding box	GPR-95
Char_GetCenter	Finds a character's center.	GPR-96
Char_GetSize	Returns the number of children	GPR-97
Char_GetTransform	Get the local transformation matrix for a character	GPR-98
Char_GetUserData	Get user-defined data pointer	GPR-99
Char_IsChild	Returns TRUE if character has a parent	GPR-100
Char_IsCulling	Returns TRUE if the character is doing bounding box culling	GPR-101
Char_IsParent	Returns TRUE if character has children	GPR-102
Char_IsVisible	Returns TRUE if the character will be rendered	GPR-103
Char_Last	Returns the last child of a character	GPR-104
Char_LookAt	Turn a character to face a certain point	GPR-105
Char_Move	Move the character in the local coordinate system.	GPR-106
Char_Next	Find the next character in a hierarchy	GPR-107
Char_Pitch	Rotate the character about the local X axis.	GPR-108
Char_Print	Prints character in ASCII SDF to standard output	GPR-109
Char_PutAfter	Insert a Character after the given character.	GPR-110
Char_PutBefore	Insert a Character before the given character.	GPR-111
Char_Remove	Removes a child from its parent	GPR-112
Char_Reset	Reset all transformations to initial conditions	GPR-113
Char_Roll	Rotate the character about the local Z axis.	GPR-114
Char_Rotate	Rotate the character in the world coordinate system	GPR-115
Char_Scale	Resize the character in the world coordinate system	GPR-116
Char_SetCulling	Enables or disables bounding box culling	GPR-117
Char_SetTransform	Set the transformation matrix	GPR-118

Char_SetUserData	Set user-defined data pointer.....	GPR-119
Char_SetVisible	Makes a character visible or invisible	GPR-120
Char_Size	Resize the character relative to the local coordinate system.....	GPR-121
Char_TotalTransform.....	Compute the total transformation matrix for this character	GPR-122
Char_Translate	Move the character in the world coordinate system.	GPR-123
Char_Turn.....	Rotate the character in the local coordinate system.	GPR-124
Char_Yaw	Rotate the character about the local Y axis.....	GPR-125
Mod_Create	Create a new model	GPR-126
Mod_GetMaterials	Get the materials array for a model.....	GPR-127
Mod_GetSurface.....	Get the surface of a model.....	GPR-128
Mod_GetTextures	Get the textures array for a model	GPR-129
Mod_SetMaterials.....	Set the materials for a model	GPR-130
Mod_SetSurface	Set the surface of a model	GPR-131
Mod_SetTextures	Set the textures for a model.....	GPR-132
Scene_Display	Display the objects in a scene.....	GPR-133
Scene_GetAmbient	Return the color of ambient light for a scene.....	GPR-134
Scene_GetBackColor	Get the background color for a scene	GPR-135
Scene_GetBound	Get the Bounding Box of a Scene.....	GPR-136
Scene_GetCamera	Return the camera associated with a scene	GPR-137
Scene_GetDynamic.....	Return the character associated with a scene	GPR-138
Scene_GetEngines	Return the array of animation engines used by a scene.....	GPR-139
Scene_GetFrameOptions	Get the current FrameOptions setting	GPR-140
Scene_GetLight	Return the default light for a scene.....	GPR-141
Scene_GetLinks.....	Return the array of links between animation engines and characters	GPR-142
Scene_GetStatic	Return the current background character	GPR-143
Scene_IsAutoAdjust	Determine if AutoAdjusting is enabled	GPR-144
Scene_IsAutoClip	Determine if AutoClipping is enabled.....	GPR-145
Scene_IsTransparent.....	Determine if scene ha a background color.....	GPR-146
Scene_IsVisible	Determine if a scene is displayed.....	GPR-147
Scene_SetAmbient	Set the color of ambient light for a scene	GPR-148
Scene_SetAutoAdjust	Enable or disable automatic aspect adjustment.....	GPR-149
Scene_SetAutoClip	Enable or disable automatic clip adjustment.....	GPR-150
Scene_SetBound	Set the Bounding Box of a Scene.....	GPR-151
Scene_SetCamera.....	Set the camera associated with a scene.....	GPR-152
Scene_SetDynamic.....	Set the character associated with a scene	GPR-153
Scene_SetEngines	Set the array of animation engines for a scene	GPR-154
Scene_SetFrameOptions.....	Set the current FrameOptions setting	GPR-155
Scene_SetLinks	Set the array of links between animation engines and characters.....	GPR-156
Scene_SetStatic.....	Set the current background character.....	GPR-157
Scene_SetTransparent.....	Make a scene transparent or not.....	GPR-158
Scene_SetVisibility	Set the visibility of a scene	GPR-159
Scene_ShowAll.....	Move the camera so all objects in a scene are visible.....	GPR-160

3**3D Pipeline Link Library Calls****box**

Box3_Around.....	Compute box around points.	GPR-163
Box3_Center	Returns the center of the bounding box.....	GPR-164
Box3_ExtendBox.....	Extend this box to enclose a second box.	GPR-165
Box3_ExtendPt	Extend this box to enclose the given array of points.....	GPR-166
Box3_ExtendPt	Extend this box to enclose the given point.....	GPR-167
Box3_IsEqual.....	Returns TRUE if the two input boxes are the same.....	GPR-168
Box3_Normalize.....	Normalize a box.	GPR-169
Box3_Print.....	Print the contents of the box [xmin, ymin, zmin : xmax, ymax, zmax]	GPR-170
Box3_Set	Initialize a bounding box from its bounds.....	GPR-171
Box3_Transform.....	Transform this box by a 4x4 matrix.	GPR-172

gp

Gfx_PipeInit	Initialize the graphics pipeline.....	GPR-173
GP_Clear.....	Clear the destination and/or the Z buffer.....	GPR-174
GP_ClearLights	Disables all lights.....	GPR-175
GP_Create	Create a graphics pipeline object which contains all the state needed to transform and render the specified geometry.	GPR-176
GP_DrawLines	Draw a connected set of lines.	GPR-177
GP_DrawPoints	Draw a set of points.....	GPR-178
GP_DrawQuadMesh.....	Draw a quad mesh.....	GPR-179
GP_DrawTriFan.....	Draw a triangle fan.....	GPR-180
GP_DrawTriList	Draw a set of triangles.	GPR-181
GP_DrawTriStrip.....	Draw a triangle strip.	GPR-182
GP_Flush	Flushes the command list buffer.....	GPR-183
GP_GetAmbient	Returns the current ambient lighting value.	GPR-184
GP_GetCullFaces	Returns the current culling operation.	GPR-185
GP_GetDestBuffer.....	Get the bitmap to be used for rendering.	GPR-186
GP_GetHiddenSurf.....	Returns the current hidden surface operation.....	GPR-187
GP_GetLight	Get the specified light information.	GPR-188
GP_GetMaterial.....	Return a pointer to the current material.....	GPR-189
GP_GetModelView	Return a pointer to the current model/view matrix.	GPR-190
GP_GetProjection	Return a pointer to the current projection matrix.....	GPR-191
GP_GetTexture.....	Return a pointer to the current texture.....	GPR-192
GP_GetTotalTransform	Return a pointer to the total transformation matrix.....	GPR-193
GP_GetViewport	Get a pointer to the current viewport.	GPR-194
GP_GetZBuffer.....	Get the bitmap to be used for Z buffering.	GPR-195
GP_IsInView	Test whether a box intersects the current view volume.	GPR-196
GP_Pop().....	Pop all the GP attributes off the stack.....	GPR-197
GP_Push().....	Push all the GP attributes onto the stack.....	GPR-198
GP_SetAmbient	Sets the current ambient lighting value.....	GPR-199
GP_SetCullFaces.....	Set the current culling operation.....	GPR-200
GP_SetDestBuffer	Specify the bitmap to be used for rendering.	GPR-201

GP_SetGState	Set the GState which a GP object should use.	GPR-202
GP_SetHiddenSurf	Set the current hidden surface operation.	GPR-203
GP_SetLight	Sets the given light in the graphics pipeline.	GPR-204
GP_SetMaterial	Sets the current material.	GPR-205
GP_SetTexture	Sets the current texture.	GPR-206
GP_SetViewport	Sets the current viewport.	GPR-207
GP_SetZBuffer	Specify the bitmap to be used for Z buffering.	GPR-208

obj

Obj_Assign	Assign a new GfxObj to the given reference.	GPR-209
Obj_Clone	Create a new, distinct copy of the given object.	GPR-210
Obj_Copy	Copies the contents of one object into another.	GPR-211
Obj_Create	Creates an object.	GPR-212
Obj_Delete	Deletes an object.	GPR-213
Obj_IsClass	Determine whether the given class is a parent class of this class.	GPR-214
Obj_Print	Prints an object.	GPR-215

surf

Surf_AddGeometry	Render a surface.	GPR-216
Surf_Create	Creates a surface.	GPR-217
Surf_Delete	Deletes a surface.	GPR-218
Surf_Display	Render a surface.	GPR-219
Surf_FindGeometry	Find's the n'th primitive added to the surface.	GPR-220
Surf_Touch	Tell the Surface that geometry has been modified transforms.	GPR-221
Trans_Add	Add source matrix to this matrix.	GPR-222
Trans_AxisRotate	Post multiply a matrix that specifies a rotation about an arbitrary axis.	GPR-223
Trans_Create	Allocate memory for a new transform object.	GPR-224
Trans_Factor	Factor a matrix into its scaling, translation and rotation components.	GPR-225
Trans_Frustum	Post multiply a matrix that specifies a perspective view volume.	GPR-226
Trans_GetData	Returns the MatrixData associated with a transform object.	GPR-227
Trans_GetInverse	Returns the inverse of a transform object.	GPR-228
Trans_Identity	Replace this matrix with identity matrix.	GPR-229
Trans_Init	Initialize matrix from a MatrixData pointer.	GPR-230
Trans_Inverse	Replace this matrix with the inverse of the source matrix.	GPR-231
Trans_LookAt	Post multiply a matrix which specified the viewer's location, direction of view and the up direction.	GPR-232
Trans_Mul	Multiply the input matrices and store in this matrix.	GPR-233
Trans_Ortho	Post multiply a matrix that specifies an orthographic projection.	GPR-235
Trans_Ortho2	Post multiply a matrix that specifies a 2d orthographic projection.	GPR-236
Trans_Perspective	Post multiply a matrix that defines a perspective projection transformation.	GPR-238
Trans_Polarview	Post multiply a matrix that defines the viewer's position in polar coordinates.	GPR-240
Trans_PostMul	Post multiply this matrix by source matrix and store result in this matrix.	GPR-242
Trans_PreMul	Pre multiply this matrix by source matrix and store result in this matrix.	GPR-243
Trans_Print	Print the transform object to standard output.	GPR-244
Trans_Rotate	Post multiply rotation matrix about coordinate axis.	GPR-245

Trans_Scale	Post multiply scaling matrix to this matrix.	GPR-247
Trans_SetData	Initialize a matrix from a 4x4 data array.	GPR-248
Trans_Subtract	Subtract source matrix from this matrix.	GPR-249
Trans_Translate	Post multiply translation matrix to this matrix.	GPR-250
Trans_Transpose	Replace this matrix with transpose of source matrix.	GPR-251

vector

Pt3_Distance	Compute distance between two points.	GPR-252
Pt3_Print	Print a Point3 to stdout.	GPR-253
Pt3_Transform	Transform an array of points (ignoring perspective).	GPR-254
Pt3_Transform	Transform a point (ignoring perspective)	GPR-255
Pt4_Print	Print a Point4 to stdout.	GPR-256
Pt4_Transform	Transform a point	GPR-257
Pt4_TransMany	Transform an array of points	GPR-258
Vec3_Add	Add two vectors	GPR-259
Vec3_Cross	Compute the cross product of two vectors	GPR-260
Vec3_Dot	Compute the dot product of two vectors	GPR-261
Vec3_Length	Compute the length of a vector	GPR-262
Vec3_LengthSquared	Compute the length of the vector squared.	GPR-263
Vec3_Negate	Negate a vector	GPR-264
Vec3_Normalize	Normalize a vector.	GPR-265
Vec3_Print	Print a Vec3_Print to stdout.	GPR-266
Vec3_Subtract	Subtract two vectors.	GPR-267
Vec3_Transform	Transform a vector (ignoring translation & perspective)	GPR-268
Vec3_TransMany	Transform an array of vectors (ignoring translation & perspective)	GPR-269

4

Font Folio Calls

CloseFont	Releases font resources.	GPR-273
CreateTextState	Creates a TextState for rendering groups of strings.	GPR-274
DeleteTextState	Deletes a TextState.	GPR-275
DrawString	Immediately draws a single string to the bitmap.	GPR-276
DrawText	Draws the text strings into the GState.	GPR-277
GetCharacterData	Gets data about a single character	GPR-278
GetStringExtent	Gets the StringExtent of a string.	GPR-279
GetTextAngle	Gets the current angle of the text in the TextState	GPR-280
GetTextExtent	Gets the TextExtent of a string.	GPR-281
GetTextPosition	Gets the current position of the text in the TextState.	GPR-282
GetTextScale	Gets the current scale value of the text in the TextState.	GPR-283
MoveText	Changes the position of text in a TextState	GPR-284
OpenFont	Loads a 3DO font file.	GPR-285
RotateText	Rotates the text in a TextState	GPR-287
ScaleText	Changes the size of text in a	GPR-288
TextState SetClipBox.	Sets the clip box for a TextState.	GPR-289
SetTextColor	Sets the color of all the text in a TextState	GPR-290
TrackTextBounds	Enables or disables the tracking of a TextState's bounding box.	GPR-291

5

Graphics Folio Calls

AddViewToViewList	Add a View to a list of Views.....	GPR-295
CloseGraphicsFolio	Terminate session with the graphics folio.....	GPR-297
CreateTEIOReq	Create an I/O request Item for use with the triangle engine device.	GPR-298
DeleteTEIOReq.....	Delete an I/O request Item procured with \f4CreateTEIOReq()\fP.	GPR-299
GfxSendCommandList.....	Send a command list to the triangle engine device.....	GPR-300
LockDisplay	Lock down changes to a Projector.....	GPR-301
ModifyGraphicsItem	Modify the characteristics of a Graphics Folio Item.	GPR-303
OpenGraphicsFolio	Initiate a session with the graphics folio.....	GPR-304
OrderViews	Depth-arrange a View relative to another View.	GPR-305
PixelAddress.....	Return the address of a given pixel.....	GPR-307
RemoveView.....	Remove a View from its ViewList.	GPR-308
UnlockDisplay.....	Release Projector lock; integrate all pending changes.....	GPR-309

6

GState (Graphics State)

GS_AllocBitmaps	Utility to allocate frame buffers and Z-buffer	GPR-313
GS_AllocLists	Allocate one or more command list buffers for graphics rendering	GPR-315
GS_BeginFrame	Notifies a GState that the next cmd. list is the first for a frame	GPR-316
GS_Create.....	Creates a GState (Graphics State) object	GPR-317
GS_Delete.....	Frees all memory associated with a GState object	GPR-318
GS_FreeBitmaps	Utility to free bitmaps and Z-buffer	GPR-319
GS_FreeLists	Free the memory used by command lists for a GState object	GPR-320
GS_GetCmdListIndex	Returns index of which cmd list buffer is in use	GPR-321
GS_GetCount	Get the current GState counter.	GPR-322
GS_GetCurListStart	Get ptr to start of the current command list buffer	GPR-323
GS_GetDestBuffer.....	Get the Item number of the bitmap being used as an output frame buffer	GPR-324
GS_GetVidSignal	Returns the signal bit which a GState is using for video synchronization	GPR-325
GS_GetZBuffer.....	Get the Item number of the bitmap being used as a Z-buffer.....	GPR-326
GS_Init	(OBSOLETE) Initializes a GState object to a default state.....	GPR-327
GS_Resume.....	Reserve block of memory in GState's current command list buffer	GPR-328
GS_SendList.....	Send a GState's current command list to the Triangle Engine	GPR-329
GS_SetDestBuffer.....	Set a bitmap as the output frame buffer for a GState	GPR-331
GS_SetList.....	Set a GState to use a new command list buffer	GPR-332
GS_SetVidSignal.....	Attach a signal to a GState object, for video synchronization	GPR-333
GS_SetZBuffer	Set a bitmap as the Z-buffer for a GState.....	GPR-334
GS_WaitIO	Wait for all pending IO to complete for a GState.....	GPR-335

7

Triangle Engine Command List Toolkit

CLT_AllocSnippet	Allocate space for the data for a CltSnippet	GPR- 339
CLT_ClearFrameBuffer.....	Clear frame buffer and/or Z-buffer via CLT cmds	GPR-340
CLT_ComputePipLoadCmdListSize	Compute size of PIP load command list	GPR-341
CLT_ComputeTxLoadCmdListSize	Compute size of texture load command list	GPR-342

CLT_CopySnippetData	Copy data portion of a cmd list snippet to specified location	GPR-343
CLT_CreatePipCommandList..	Creates a cmd list snippet to load a PIP	GPR-344
CLT_CreateTxLoadCommandList	Creates a cmd list snippet to load a texture	GPR-345
CLT_FreeSnippet	Free memory allocated for a CltSnippet's data.....	GPR-347
CLT_InitSnippet.....	Initializes a CltSnippet structure	GPR-348
CLT_SetSrcToCurrentDest.....	Set the Dest Blender so that src blends will occur with cur frame buffer..	GPR-349

Globals

CltEnableTextureSnippet.....	Global var. used to enable texturing.....	GPR-350
CltNoTextureSnippet.....	Global var. used to disable texturing.....	GPR-351

Volume 3:

3DO M2 Tools For Sound Design

Preface

About This Document.....	TSD-xiii
Audience.....	TSD-xiii
How This Document Is Organized	TSD-xiii
Typographical Conventions	TSD-xiv

1

Music and Sound Effects for 3DO Titles

Introduction.....	TSD-1
Chapter Overview	TSD-1
Resource Allocation and Planning	TSD-2
RAM-Resident vs. ROM-Resident Sound	TSD-2
Spooled vs. Streamed Sound	TSD-2
Sound File Characteristics	TSD-3
Audio Production.....	TSD-4
Sampling Sound Effects	TSD-4
Working With Sampled Music	TSD-5
Creating Algorithmic Sound Effects	TSD-6
Incorporating Sound Effects Into Your Title	TSD-6
Playing MIDI Scores	TSD-6
Streaming Audio Data With the 3DO DataStreamer	TSD-7
Aesthetic Considerations	TSD-7
Tweak, Tweak, Tweak!	TSD-7
Project Management.....	TSD-8

2

Installing and Launching ARIA

Introduction.....	TSD-11
Chapter Overview	TSD-11
Hardware and Software Configuration.....	TSD-12
Launching and Setting Up ARIA	TSD-12
Troubleshooting	TSD-13
Setup for Working With MIDI	TSD-13
Files and Folder Setup	TSD-13
Required Files for Real-time MIDI	TSD-13
Setting Up for MIDI	TSD-14
Drag and Drop ARIA Installation.....	TSD-14

3

Creating MIDI Projects With ARIA

Introduction.....	TSD-17
Chapter Overview	TSD-17
Preparing Your Environment	TSD-18

MIDI Files and PIMap Files	TSD-18
Working With PIMaps	TSD-18
Playing a MIDI File on the 3DO Station	TSD-20
Working With General MIDI	TSD-22
Working With Real-time MIDI	TSD-23
Tips and Tricks	TSD-23
MIDI Project Interface Tips	TSD-23
MIDI Tricks	TSD-24

4

SquashSnd

Introduction	TSD-25
SquashSnd-Compatible File Types	TSD-25
SquashSnd Parameters	TSD-26
Caveats	TSD-27

5

3SF Overview

Introduction	TSD-29
The 3DO Score File Format	TSD-30
3SF Architecture	TSD-30
Before 3SF	TSD-30
Method 1: Play a Sample in Memory	TSD-31
Method 2: Spool a Sample From Disk	TSD-31
Method 3: Weave a Sample Into a Stream	TSD-31
Method 4: Play a Synthetic Patch in the DSP	TSD-31
Method 5: Play a MIDI File	TSD-31
Sound Design With 3SF	TSD-32
Score File Creation	TSD-32
From ARIA	TSD-32
MakeScore Tool	TSD-33
Score File Playbackscore	TSD-33
From ARIA	TSD-33
PlayScore	TSD-33
From a C or C++ Program	TSD-33
.....	TSD-33

6

3SF Architecture

Overview	TSD-35
Creating a 3DO Score File	TSD-36
Playing a 3DO Score File	TSD-37

7

3SF User's Guide

Introduction	TSD-39
Why Use 3SF	TSD-40

The 3SF Tool Set	TSD-40
MakeScore	TSD-41
DumpIFF	TSD-42
PlayScore	TSD-43
PlayScore.lib.....	TSD-43
ScorePlayer	TSD-43
ErrorProc	TSD-44
StartReadingAndPlaying	TSD-44
StartReading	TSD-44
Play, Pause, Stop	TSD-45

8

Essential Features of SoundHack

Introduction.....	TSD-47
Chapter Overview	TSD-47
Installing SoundHack.....	TSD-48
Converting a Sound File to Compressed Format.....	TSD-48
Loading a Sound File	TSD-48
Playing the Current Sound File	TSD-49
Converting the Current Sound File to Compressed Format	TSD-49
Reading/Writing Raw Files.....	TSD-49
Changing Header Information.....	TSD-50
Header Change Command	TSD-50
Loops & Markers Command	TSD-51

9

SoundHack User's Guide

Introduction.....	TSD-53
Chapter Overview	TSD-53
The File Menu	TSD-54
Open Command	TSD-54
Open Any Command	TSD-55
Close Command	TSD-55
Save A Copy Command	TSD-56
Listen to AIFF File command	TSD-56
Import SND Resource Command	TSD-56
Export SND Resource Command	TSD-56
Quit Command	TSD-56
The Edit Menu	TSD-56
The Hack Menu.....	TSD-56
Binaural Filter	TSD-57
Processing a File With the Binaural Filter	TSD-58
Convolution Command	TSD-58
Gain Change Command	TSD-61
Mutation Command	TSD-61
Phase Vocoder Command	TSD-64
Spectral Dynamics Command	TSD-66

Varispeed Command	TSD-68
The Control Menu	TSD-70
Show Output Command	TSD-70
Show Spectrum Command	TSD-70
Pause Process Command	TSD-70
Continue Process Command	TSD-70
Stop Process Command	TSD-70
Bibliography	TSD-71

10

AudioThing

Introduction	TSD-73
System Requirements and Options	TSD-73
Installation	TSD-74
Using AudioThing	TSD-74

11

Tips, Tricks, and Troubleshooting

Available audio tools	TSD-75
SoundHack	TSD-75
SquashSound	TSD-75
ARIA	TSD-75
MakePatch	TSD-76
AIFF Files	TSD-76
AIFF files in Examples folder	TSD-76
AIFF and compression	TSD-76
Preparing Audio	TSD-76
Mastering audio	TSD-76
Editing Synthesized Audio	TSD-76

A

AIFF Samples

Overview	TSD-79
File names	TSD-80
Auditioning samples from the Sound Designer II Edit window	TSD-80
Folder contents	TSD-81
Samples in GMPercussion22k and GMPercussion44k	TSD-81
Samples in the PitchedL folder	TSD-83
Samples in the PitchedLR folder	TSD-83
Samples in the Unpitched folder	TSD-83

B

DumpIFF Sample Output

Introduction	TSD-87
--------------------	--------

C

3SF File Format Specification

Introduction	TSD-89
File Types and Extensions.....	TSD-89
Data Types.....	TSD-90
File Structure	TSD-92
Chunk	TSD-92
FORM Chunk	TSD-93
Syntax Definitions	TSD-94
File Version Chunk	TSD-95
Sound Set	TSD-95
Score Items	TSD-96
Licks	TSD-97
Axes	TSD-97
Item Numbers and Item Names	TSD-97
Set Header	TSD-99
Item Header	TSD-99
The Global Numeric Index	TSD-100
Specifying a Score Item or a Sound Set	TSD-101
The Name Index	TSD-102
Dependency Lists	TSD-104
MIDI Context Format	TSD-104
MIDI Event Format	TSD-105

3DO M2 Audio And Music Programmer's Guide

Preface

About this Book	MPG-xvii
About the Audience	MPG-xvii
How this Book Is Organized.....	MPG-xvii
Related Documentation	MPG-xviii
About the Code Examples.....	MPG-xix
Typographical Conventions	MPG-xix

1

How To Write Audio Software

How Do I Make a Simple Sound?	MPG-2
How Do I Play an Audio Sample?	MPG-2
How Do I Determine When a Sample or Envelope Finishes Playing on a Specific Instrument?	MPG-3
How Do I Play a Long Sound File?	MPG-3
How Do I Play a Musical Score?.....	MPG-4
How Do I Manage Multiple Sound Effects?	MPG-5

How Do I Create Complex Sound Effects?	MPG-6
How Do I Make a Sound Fade Out Smoothly without Popping?	MPG-6

2

Understanding 3DO Audio

What is 3DO Audio?	MPG-8
Audio Hardware	MPG-8
The CD Player	MPG-9
Audio Software	MPG-10
Designing Sound with the Audio Folio	MPG-10
Playing Notes	MPG-13
Using the Music Library	MPG-15

3

Playing a Synthetic Sound

Introduction	MPG-19
The Process of Using Instruments	MPG-20
Accessing the Audio Folio	MPG-21
Instruments and Templates	MPG-21
Template	MPG-21
Instrument	MPG-22
Kinds of Instrument Templates	MPG-22
Standard Instrument Templates	MPG-22
Instrument Ports	MPG-25
Preparing Instruments	MPG-26
Loading a Standard Instrument Template	MPG-26
Creating a Mixer Template	MPG-27
Creating an Instrument	MPG-28
Calculation Rates	MPG-29
Instrument Resources	MPG-30
Loading a Standard Instrument	MPG-30
Connecting Instruments	MPG-31
Connecting One Instrument to Another	MPG-31
Disconnecting One Instrument From Another	MPG-33
Attachments	MPG-33
Playing Instruments	MPG-34
Starting an Instrument	MPG-34
Releasing an Instrument	MPG-37
Stopping an Instrument	MPG-38
Cleaning Up When Finished	MPG-38
Introducing the Audio Clock	MPG-40
Reading the Global Audio Clock	MPG-40

4

Playing a Sampled Sound

Introduction	MPG-43
Sampled Sound	MPG-44
Loading and Attaching Samples	MPG-46
Loading Samples from Disk	MPG-46
Sample Loops	MPG-50
Sample Trigger Points	MPG-51
Attaching a Sample to an Instrument	MPG-51
The Attachment Item	MPG-53
Attaching Multisamples to an Instrument	MPG-53
Detaching a Sample from an Instrument	MPG-54
Deleting a Sample	MPG-54
Debugging a Sample	MPG-54
Example Program	MPG-55
Modifying Attachments	MPG-55
Setting an Attachment's Attributes	MPG-55
Specifying a Start Point	MPG-55
Setting a Start-Independent Attachment	MPG-56
Creating an Instrument-Stopping Attachment	MPG-56
Independently Controlling Attachments	MPG-57
Starting an Attachment	MPG-57
Releasing an Attachment	MPG-57
Stopping an Attachment	MPG-57
Linking Attachments	MPG-58

5

Controlling Sound Parameters

Introduction	MPG-61
Digital Signal Processing Signal Types	MPG-61
Audio and Control Signals	MPG-61
Signal Flow Between Connected Instruments	MPG-62
Control Signal Arithmetic	MPG-63
Knobs and Probes	MPG-64
Type Casting for Knobs and Probes	MPG-66
Handling Knobs	MPG-67
Finding Knobs	MPG-67
Creating a Knob	MPG-69
Finding Knob Parameters	MPG-70
Setting Knob Values	MPG-70
Reading Knob Values	MPG-71
Controlling Knobs with Generic Values	MPG-71
Deleting a Knob	MPG-73
Creating and Attaching Envelopes	MPG-74
Envelope Properties	MPG-74
Creating an Envelope	MPG-83

Attaching an Envelope to an Instrument	MPG-83
Detaching an Envelope from an Instrument	MPG-85
Deleting an Envelope	MPG-85

6

Advanced Audio Folio Usage

Introduction	MPG-87
Reading and Changing Audio Item Attributes	MPG-87
Reading Audio Item Characteristics	MPG-88
Setting Audio Item Characteristics	MPG-88
Reading Instrument Output	MPG-89
Creating the Probe	MPG-89
Using Probes	MPG-89
Tuning Instruments	MPG-90
Creating a Tuning	MPG-90
Applying a Tuning	MPG-92
Deleting a Tuning	MPG-92
Bending Pitch	MPG-93
Adding Reverberation	MPG-94
A Delay Line Overview	MPG-94
Creating a Delay Line	MPG-95
Deleting a Delay Line	MPG-96
Connecting a Delay Line to Create Reverberation	MPG-96
Using a Delay Line for Oscilloscope Data	MPG-99
Audio Clocks.....	MPG-99
Creating a Custom Audio Clock	MPG-99
Setting the Clock Rate	MPG-100
Function Calls	MPG-100
Tuning	MPG-100
Adding Reverberation	MPG-101
Audio Clocks	MPG-101

7

Patch Templates

Introduction	MPG-103
Patch Compiler	MPG-104
Binary Patch File Loader	MPG-104
Makepatch	MPG-104
Patch Examples	MPG-104
Reverberation Examples	MPG-110

8

3D Sound Spatialization

Introduction	MPG-117
Chapter Overview	MPG-117
Sound Cues	MPG-117
Amplitude Panning	MPG-118

Delay Panning	MPG-118
Filtering	MPG-118
Doppler	MPG-118
Distance Factor	MPG-119
Directionality	MPG-119
Locating 3D Sound Examples	MPG-120
Using 3D Sound	MPG-120
Setting up 3D Sounds	MPG-121
Pseudo Code example of 3D Sound	MPG-122
Starting 3D Sounds	MPG-122
Moving 3D Sounds	MPG-123
Deleting 3D Sounds	MPG-124

9 Sound Player

An Overview of the Sound Player	MPG-125
Simple Sound Player Example	MPG-126
Players, Sounds, and Markers	MPG-128
Players	MPG-128
Sounds	MPG-128
Markers	MPG-128
Examples of Looping and Branching	MPG-129
Decision Functions	MPG-130
Decision Functions and Actions	MPG-132
Example Decision Function	MPG-132
Rules for Decision Functions	MPG-133
Caveats	MPG-134
For More Information	MPG-134
Function Calls	MPG-134
Player Management Function Calls	MPG-134
Sound Management Function Calls	MPG-135
Marker Management Function Calls	MPG-135
Decision Function Calls	MPG-136
Debug Function Calls	MPG-137
Constants	MPG-137

10 Using the Sound Spooler

How the Sound Spooler Works	MPG-140
How to Use the Sound Spooler.....	MPG-140
An Example.....	MPG-141
Convenience Functions.....	MPG-145
ssplSpoolData()	MPG-145
ssplPlayData()	MPG-146
Sound Spooler Function Calls.....	MPG-146
Convenience calls	MPG-147
SoundSpooler management calls	MPG-147

SoundBufferNode management calls	MPG-147
Low level calls	MPG-148
Callback routine	MPG-148

11

Playing MIDI Scores

A Brief MIDI Review.....	MPG-151
MIDI Channels	MPG-151
Channel Messages	MPG-152
MIDI Score Playback	MPG-152
Creating an Internal MIDI Environment.....	MPG-153
Creating a Virtual MIDI Device	MPG-153
Importing a MIDI Score	MPG-155
Providing MIDI Playback Functions	MPG-156
Setting the Audio Clock Speed to Control Tempo	MPG-157
An Overview of the MIDI Score-Playing Process	MPG-157
Setting Up.....	MPG-158
Creating a MIDI Environment.....	MPG-158
Setting Voice and Program Limits	MPG-158
Creating a Score Context	MPG-159
Setting PIMap Entries	MPG-160
Setting Up a Mixer	MPG-164
Importing a MIDI Score	MPG-165
Declaring a MIDIFileParser Data Structure	MPG-165
Creating a Juggler Object	MPG-165
Loading the Score	MPG-165
Specifying the User Context	MPG-166
The Interpreter Procedure	MPG-167
Setting the Tempo	MPG-168
Setting the Audio Clock Rate	MPG-168
Playing the MIDI Score.....	MPG-168
Dynamic Voice Allocation	MPG-170
Freeing Instruments Created by Voice Allocation	MPG-171
Using MIDI Functions.....	MPG-172
Interpreting and Executing a MIDI Message	MPG-172
Starting and Releasing a Note	MPG-173
Directly Starting and Releasing an Instrument	MPG-174
Changing a Program	MPG-175
Setting Channel Panning and Volume	MPG-175
Bending Pitch	MPG-176
Changing Characteristics During Playback	MPG-178
Cleaning Up	MPG-179
Creating a MIDI Score for Playback.....	MPG-180
Primary Data Structures	MPG-181

Function Calls	MPG-181
MIDI Environment Calls	MPG-181
MIDI Score Calls	MPG-182
MIDI Playback Calls	MPG-182

12

Creating and Playing Juggler Objects

Object-Oriented Programming Concepts	MPG-184
Objects	MPG-184
Methods and Messages	MPG-184
Object Variables	MPG-185
Classes and Instances	MPG-185
Creating New Classes	MPG-186
Managing Juggler Objects	MPG-186
Initializing the Juggler	MPG-186
Defining a New Class	MPG-187
Creating an Object	MPG-187
Destroying an Object	MPG-187
Checking an Object's Validity	MPG-188
Default Juggler Classes	MPG-188
The Jugglee Class	MPG-188
The Sequence Class	MPG-189
The Collection Class	MPG-190
Creating Sequences and Collections	MPG-190
Creating a Sequence	MPG-191
Creating a Collection	MPG-195
Sending Messages to Sequences and Collections	MPG-198
Calling Methods Directly	MPG-198
Message Macros	MPG-198
Messages for Sequences	MPG-198
Messages for Collections	MPG-201
Playing Sequences and Collections	MPG-203
A Juggler Operational Overview	MPG-203
The Juggler Process	MPG-205
Bumping the Juggler	MPG-205
Terminating the Juggler	MPG-206
An Example Program	MPG-206
Function Calls and Method Macros	MPG-210
Juggler Control Calls	MPG-210
Object Management Calls	MPG-211
Method Macros	MPG-211
Object-Defining Tag Arguments	MPG-211

13 Tips and Techniques

DSP Resources.....	MPG-214
Dynamic Voice Allocation	MPG-214
Audio Samples.....	MPG-214
Available RAM for Sampling	MPG-214
Looping Stereo and Mono Sound	MPG-214
Translating PC VOX files	MPG-214
Loading Multiple Sound Files	MPG-215
Playing Scores.....	MPG-215
Streaming Audio versus Score Playback	MPG-215
Score Files	MPG-216
MIDI	MPG-217
Playing Red Book Audio	MPG-217
Timing	MPG-217
DSP Time versus System Time	MPG-217
Multi-thread Timers	MPG-218
Filters.....	MPG-218
Modulating Selected Frequency Ranges	MPG-218
Miscellaneous.....	MPG-218
Allocating Signals	MPG-218

14 Beep Folio

Differences Between Beep Folio and Audio Folio	MPG-219
Reasons for Choosing Audio Folio	MPG-221
Reasons for Choosing Beep Folio	MPG-221
What Beep Folio Doesn't Provide	MPG-221
Beep Folio Vocabulary	MPG-222
Machine	MPG-222
Voice	MPG-222
Channel	MPG-222
Parameter	MPG-222
Steps in Using the Beep Folio	MPG-222
Other Considerations	MPG-223
Mixing	MPG-223
Cache Coherency	MPG-223
Knowing when a sample is finished playing	MPG-223

3DO M2 Audio Programmer's Reference

1

Audio Folio Calls

--Audio Port-Types--	Audio port type descriptions.....	APR-3
--Audio Signal-Types--	Audio signal type descriptions.....	APR-4

Attachment

Create Attachment	Creates an Attachment between a Sample or Envelope to an Instrument or Template.	APR-6
Delete Attachment	Undoes Attachment between Sample or Envelope and Instrument or Template.	APR-8
GetAttachments	Returns list of Attachments to a sample or envelope hook.	APR-9
GetNumAttachments	Returns the number of Attachments to a sample or envelope hook.....	APR-10
LinkAttachments	Connects Sample Attachments for sequential playback.	APR-11
MonitorAttachment	Monitors an Attachment, sends a Cue at a specified point.....	APR-13
ReleaseAttachment	Releases an Attachment.....	APR-14
StartAttachment.....	Starts an Attachment.	APR- 15
StopAttachment.....	Stops an Attachment.	APR-16
WhereAttachment	Returns the current playing location of an Attachment.	APR-17

Cue

CreateCue	Creates an audio Cue.	APR-18
DeleteCue	Deletes an audio Cue	APR-19
GetCueSignal	Returns a signal mask of a Cue	APR-20

Envelope

CreateEnvelope	Creates an Envelope.....	APR-21
DeleteEnvelope	Deletes an Envelope.....	APR-22

Instrument

AbandonInstrument	Makes an Intrument available for adoption from Template.	APR-23
AdoptInstrument.....	Adopts an abandoned Intrument from a Template.....	APR-24
BendInstrumentPitch.....	Bends an Instrument' output pitch up or down by a specified amount.	APR-25
ConnectInstrumentParts.....	Patches the output of an Instrument to the input of another Intrument.	APR-26
ConnectIntruments	Patches the output of an Intrument to the output of another Intrument	APR-28
CreateIntrument.....	Allocates an Instrument from an Intrument Template.....	APR-29
DeleteIntrument	Frees an Intrument allocated by CreateIntrument().	APR-31
DisconnectInstrumentParts	Breaks a connection made by ConnectIntrumentParts().	APR-32
DumplInstrumentResourceInfo	Prints out Intrument resource usage information.	APR-33
GetInstrumentPortInfoByIndex	Looks up Instrument port by index and returns port information.....	APR-34
GetInstrumentPortInfoByName	Looks up Intrument port by name and returns port information.	APR-36
GetInstrumentResourceInfo ...	Get audio resource usage information for an Instrument.	APR-38
GetNumInstrumentPorts.....	Returns the number of ports of an Instrument.	APR-40
LoadInsTemplate	Loads a standard DSP Instrument \f4Template\fP.	APR-41
LoadInstrument	Loads a DSP instrument \f4Template\fP and creates an \f4Instrument\fP from it in one call.	APR-42
PauseInstrument	Pauses an Instrument's playback.	APR-44

ReleaseInstrument	Instruct an Instrument to begin to finish (Note Off).....	APR-45
ResumeInstrument	Resumes playback of a paused instrument.....	APR-46
StartInstrument	Begins playing an \f4Instrument\fP (Note On).	APR-47
StopInstrument.....	Abruptly stops an \f4Instrument\fP.....	APR-50
UnloadInsTemplate	Unloads an instrument \f4Template\fP loaded by \f4LoadInsTemplate()\fP.	APR-51
UnloadInstrument.....	Unloads an instrument loaded with \f4LoadInstrument()\fP.	APR-52

Knob

CreateKnob	Gain direct access to one of an \f4Instrument\fP's \f4Knob\fPs.....	APR-53
DeleteKnob	Deletes a \f4Knob\fP.	APR-55
ReadKnob	Reads the current value of a single-part \f4Knob\fP.....	APR-56
ReadKnobPart.....	Reads the current value of a \f4Knob\fP.....	APR-57
SetKnob	Sets the value of a single-part \f4Knob\fP.	APR-58
SetKnobPart.....	Sets the value of a \f4Knob\fP.....	APR-59

Miscellaneous

CloseAudioFolio	Closes the audio folio.	APR-60
EnableAudioInput.....	Enables or disables audio input.	APR-61
GetAudioFolioInfo	Get system-wide audio settings.	APR-62
GetAudioFrameCount.....	Gets count of audio frames executed.....	APR-64
GetAudioItemInfo.....	Gets information about an audio item.....	APR-65
GetAudioResourceInfo.....	Get information about available audio resources..	APR-66
OpenAudioFolio	Opens the audio folio.....	APR-68
SetAudioItemInfo.....	Sets parameters of an audio item..	APR-69

Mixer

CalcMixerGainPart.	Returns the mixer gain knob part number for a specified input and output.....	APR-70
CreateMixerTemplate	Creates a custom mixer \f4Template\fP.....	APR-71
DeleteMixerTemplate.	Deletes a mixer Template created by \f4CreateMixerTemplate()\fP	APR-73
MakeMixerSpec.	Makes a MixerSpec for use with \f4CreateMixerTemplate()\fP.	APR-74
MixerSpecToFlags.	Extracts mixer flags from MixerSpec.....	APR-76
MixerSpecToNumIn.	Extracts number of inputs from MixerSpec.	APR-77
MixerSpecToNumOut.....	Extracts number of outputs from MixerSpec.....	APR-78

Probe

CreateProbe.	Creates a \f4Probe\fP for an output of an \f4Instrument\fP.....	APR-79
DeleteProbe.	Deletes a \f4Probe\fP.....	APR-80
ReadProbe	Reads the value of a single-part \f4Probe\fP.....	APR-81
ReadProbePart.....	Reads the value of a \f4Probe\fP.....	APR-82

Sample

CreateDelayLine	Creates a Delay Line \f4Sample\fP for echoes and reverberations.	APR-83
CreateSample.....	Creates a \f4Sample\fP.....	APR-85
DebugSample.	Prints \f4Sample\fP information for debugging.....	APR-86
DeleteDelayLine.	Deletes a delay line \f4Sample\fP.....	APR-87
DeleteSample.....	Deletes a \f4Sample\fP.....	APR-88

Signal

ConvertAudioSignalToGeneric	Converts audio signal value of specified signal type to generic signal value.	APR-89
ConvertGenericToAudioSignal	Converts generic signal value to specified audio signal type.	APR-90
GetAudioSignalInfo	Get information about audio signal types.	APR-91

Timer

AbortTimerCue	Cancels a timer request enqueued with \f4SignalAtAudioTime()\fP	APR-93
AudioTimeLaterThan	Compare two AudioTime values with wraparound.	APR-94
AudioTimeLaterThanOrEqual	Compare two AudioTime values with wraparound.	APR-95
CompareAudioTimes	Compare two AudioTime values with wraparound.	APR-96
CreateAudioClock	Creates an \f4AudioClock\fP	APR-97
DeleteAudioClock	Deletes an \f4AudioClock\fP	APR-98
GetAudioClockDuration	Asks for the duration of an \f4AudioClock\fP tick in frames.	APR-99
GetAudioClockRate	Asks for \f4AudioClock\fP rate in Hertz.	APR-100
GetAudioDuration	Asks for the duration of an AF_GLOBAL_CLOCK tick in frames.	APR-101
GetAudioTime	Reads AF_GLOBAL_CLOCK time.	APR-102
ReadAudioClock	Reads AudioTime from an AudioClock.	APR-103
SetAudioClockDuration	Changes duration of \f4AudioClock\fP tick.	APR-104
SetAudioClockRate	Changes the rate of the \f4AudioClock\fP	APR-105
SignalAtAudioTime	Requests a wake-up call at a given time.	APR-106
SignalAtTime	Requests a wake-up call at a given time from AF_GLOBAL_CLOCK.	APR-107
SleepUntilAudioTime	Enters wait state until \f4AudioClock\fP reaches a specified AudioTime.	APR-108
SleepUntilTime	Enters wait state until AF_GLOBAL_CLOCK reaches a specified AudioTime.	APR-110

Trigger

ArmTrigger	Assigns a \f4Cue\fP to an \f4Instrument\fP's Trigger.	APR-111
DisarmTrigger	Remove previously assigned \f4Cue\fP from an \f4Instrument\fP's Trigger.	APR-113

Tuning

Convert12TET_FP	Converts a pitch bend value in semitones and cents into a float value.	APR-114
CreateTuning	Creates a \f4Tuning\fP Item.	APR-115
DeleteTuning	Deletes a \f4Tuning\fP	APR-117
TuneInsTemplate	Applies the specified \f4Tuning\fP to an instrument \f4Template\fP	APR-118
TuneInstrument	Applies the specified \f4Tuning\fP to an \f4Instrument\fP	APR-119

2

Audio Patch File Folio Calls

--Patch-File-Overview--	Overview of the FORM 3PCH binary patch file format.	APR-123
EnterForm3PCH	Standard FORM 3PCH entry handler.	APR-126
ExitForm3PCH	Standard FORM 3PCH exit handler.	APR-127
LoadPatchTemplate	Loads a binary patch file (FORM 3PCH).	APR-129
UnloadPatchTemplate	Unloads a Patch \f4Template\fP loaded by \f4LoadPatchTemplate()\fP.	APR-131

3**Audio Patch Folio Calls**

CreatePatchTemplate	Constructs a custom Patch \f4Template\fP from simple Instrument Templates.	APR-135
DeletePatchTemplate	Deletes a custom Instrument Template created by \f4CreatePatchTemplate()\fP	APR-137
DumpPatchCmd	Prints out a PatchCmd	APR-138
DumpPatchCmdList	Prints out a PatchCmd list.	APR-139
NextPatchCmd	Finds the next \f4PatchCmd\fP in a PatchCmd list.	APR-140

PatchCmd

PatchCmd	Command set used by \f4CreatePatchTemplate()\fP	APR-142
PATCH_CMD_ADD_TEMPLATE	Adds an instrument \f4Template\fP to patch.	APR-143
PATCH_CMD_CONNECT	Connects patch blocks and ports to one another.	APR-144
PATCH_CMD_DEFINE_KNOB	Defines a patch knob.	APR-146
PATCH_CMD_DEFINE_PORT	Defines a patch input or output port.	APR-147
PATCH_CMD_END	Marks the end of a PatchCmd list.	APR-148
PATCH_CMD_EXPOSE	Exposes a patch port.	APR-149
PATCH_CMD_JUMP	Links to another list of PatchCmds.	APR-150
PATCH_CMD_NOP	Skip this command.	APR-151
PATCH_CMD_SET_COHERENCE	Controls signal phase coherence along internal patch connections.	APR-152
PATCH_CMD_SET_CONSTANT	Sets a block input or knob to a constant.	APR-153

PatchCmdBuilder

AddTemplateToPatch	Adds a \f4PATCH_CMD_ADD_TEMPLATE\fP PatchCmd to PatchCmdBuilder.	APR- 154
ConnectPatchPorts	Adds a \f4PATCH_CMD_CONNECT\fP PatchCmd to PatchCmdBuilder.	APR-155
CreatePatchCmdBuilder	Creates a PatchCmdBuilder (convenience environment for constructing a \f4PatchCmd\fP List)	APR-157
DefinePatchKnob	Adds a \f4PATCH_CMD_DEFINE_KNOB\fP PatchCmd to PatchCmdBuilder.	APR-159
DefinePatchPort	Adds a \f4PATCH_CMD_DEFINE_PORT\fP PatchCmd to PatchCmdBuilder.	APR-160
DeletePatchCmdBuilder	Deletes a PatchCmdBuilder created by \f4CreatePatchCmdBuilder()\fP. ..	APR-161
ExposePatchPort	Adds a \f4PATCH_CMD_EXPOSE\fP PatchCmd to PatchCmdBuilder.	APR-162
GetPatchCmdBuilderError	Returns error code from a failed PatchCmd constructor.	APR-163
GetPatchCmdList	Returns PatchCmd list from a PatchCmdBuilder.	APR-164
SetPatchCoherence	Adds a \f4PATCH_CMD_SET_COHERENCE\fP PatchCmd to PatchCmdBuilder.	APR-165
SetPatchConstant	Adds a \f4PATCH_CMD_SET_CONSTANT\fP PatchCmd to PatchCmdBuilder.	APR-166

4

Beep Folio Calls

ConfigureBeepChannel	Configure a DMA channel.	APR- 169
GetBeepTime	Return the current Beep time.	APR-170
LoadBeepMachine	Load the Beep Machine DSP program into DSP.	APR-171
SetBeepChannelData.....	Specify data for a DMA channel.	APR-172
SetBeepChannelDataNext	Specify data to play after current data finishes.	APR-173
SetBeepParameter	Set a global Beep parameter.	APR-175
SetBeepVoiceParameter	Set a Beep voice parameter.	APR-176
StartBeepChannel	Start a DMA channel.	APR-177
StopBeepChannel	Stop a DMA channel.	APR-178
UnloadBeepMachine.....	Unload the Beep Machine from the DSP.	APR-179

Machines

basic.bm	Basic Beep Machine.	APR-180
----------------	--------------------------	---------

5

DSP Instruments

--DSP-Instrument-Overview--	Overview of DSP Instrument documentation.	APR-185
Mixer	General description of custom mixer \4Template\4Ps built by \4CreateMixerTemplate()\4P.....	APR-187

Accumulator

add_accum.dsp	Adds signed signal to DSP accumulator.	APR-189
input_accum.dsp	Loads the accumulator from an input.	APR-190
multiply_accum.dsp	Multiplies accumulator by a signed signal.	APR-191
output_accum.dsp.....	Stores accumulator to an output.	APR-192
subtract_accum.dsp	Subtracts accumulator from input.	APR-193
subtract_from_accum.dsp.....	Subtracts input from accumulator.	APR-194

Arithmetic

add.dsp.....	Adds two signed signals.	APR-195
expmod_unsigned.dsp	Exponential modulation of an unsigned signal.	APR-196
latch.dsp	Passes its input to output if gate held above zero.	APR-197
maximum.dsp	Picks the maximum of two input signals.	APR-198
minimum.dsp	Picks the minimum of two input signals.	APR-199
multiply.dsp	Multiplies two signed input signals (ring modulator).	APR-200
multiply_unsigned.dsp	Multiplies two unsigned signals.	APR-201
schmidt_trigger.dsp	Comparator with hysteresis and trigger.	APR-202
subtract.dsp.....	Returns the difference between two signed signals.	APR-203
timesplus.dsp	Single-operation multiply and accumulate ($A*B+C$).	APR-204
timesplus_noclip.dsp.....	Single-operation, unclipped multiply and accumulate ($A*B+C$).	APR-205
times_256.dsp.....	Fast multiplication by 256.	APR-206

Control_Signal

envelope.dsp	Interpolate a segment of an \4Envelope\4P.....	APR-207
envfollower.dsp	Tracks the contour of a signal.	APR-208

integrator.dsp	Integrates an input signal (ramp generator).	APR-209
pulse_lfo.dsp	Pulse wave Low-Frequency Oscillator.	APR-211
randomhold.dsp	Generates random values and holds them.	APR-212
rednoise_lfo.dsp	Red noise LFO generator.	APR-213
slew_rate_limiter.dsp	Slew rate limiter.	APR-214
square_lfo.dsp	Square wave Low-Frequency Oscillator.	APR-215
triangle_lfo.dsp	Triangle wave Low-Frequency Oscillator.	APR-216

Diagnostic

benchmark.dsp	Outputs current DSPP tick count.	APR-217
---------------------	----------------------------------	---------

Effects

cubic_amplifier.dsp	Non-linear amplifier (distortion effect).	APR-218
deemphcd.dsp	CD de-emphasis filter.	APR-219
delay4.dsp	Delays an input signal by up to 4 frames.	APR-220
delay_f1.dsp	Sends mono input to a delay line.	APR-221
delay_f2.dsp	Sends stereo input to a delay line.	APR-222
depopper.dsp	Helps eliminate pops when switching sounds.	APR-223
filter_1o1z.dsp	First Order, One Zero filter.	APR-224
filter_3d.dsp	Sound spatialization filter.	APR-225
svfilter.dsp	State-variable digital filter.	APR-226

Line_In_And_Out

line_in.dsp	Taps stereo audio line in.	APR-228
line_out.dsp	Adds to stereo signal to send to audio line out.	APR-229
tapoutput.dsp	Taps accumulated stereo line out signal.	APR-230

Sampled_Sound

sampler_16_f1.dsp	Fixed-rate mono 16-bit sample player.	APR-231
sampler_16_f2.dsp	Fixed-rate stereo 16-bit sample player.	APR-232
sampler_16_v1.dsp	Variable-rate mono 16-bit sample player.	APR-233
sampler_16_v2.dsp	Variable-rate stereo 16-bit sample player.	APR-234
sampler_8_f1.dsp	Fixed-rate mono 8-bit sample player.	APR-235
sampler_8_f2.dsp	Fixed-rate stereo 8-bit sample player.	APR-236
sampler_8_v1.dsp	Variable-rate mono 8-bit sample player.	APR-237
sampler_8_v2.dsp	Variable-rate stereo 8-bit sample player.	APR-238
sampler_adp4_v1.dsp	Variable-rate mono sample player with ADPCM Intel/DVI 4:1 decompression.	APR-239
sampler_cbd2_f1.dsp	Fixed-rate mono sample player with CBD2 2:1 decompression.	APR-240
sampler_cbd2_f2.dsp	Fixed-rate stereo sample player with CBD2 2:1 decompression.	APR-241
sampler_cbd2_v1.dsp	Variable-rate mono sample player with CBD2 2:1 decompression.	APR-242
sampler_cbd2_v2.dsp	Variable-rate stereo sample player with CBD2 2:1 decompression.	APR-243
sampler_drift_v1.dsp	Variable-rate mono 16-bit sample player with drift output (suitable for flanging).	APR-244
sampler_raw_f1.dsp	Fixed-rate mono 16-bit sample player without amplitude scaling.	APR-245
sampler_sqs2_f1.dsp	Fixed-rate mono sample player with SQS2 2:1 decompression.	APR-246
sampler_sqs2_v1.dsp	Variable-rate mono sample player with SQS2 2:1 decompression.	APR-247

Sound_Synthesis

chaos_1d.dsp	One-dimensional chaotic function generator (the logistic map).....	APR-248
impulse.dsp	Impulse waveform generator.	APR-249
noise.dsp	White noise generator.....	APR-250
pulse.dsp	Pulse wave generator.....	APR-251
rednoise.dsp	Red noise generator.....	APR-252
sawtooth.dsp	Sawtooth wave generator.	APR-253
square.dsp	Square wave generator.	APR-254
triangle.dsp	Triangle wave generator.....	APR-255

6

Music Link Library Calls

ATAG

--ATAG-File-Format--	Simple audio object file format for Samples, Envelopes, Delay lines, and Tunings.	APR-259
AudioTagHeaderSize	Returns size of AudioTagHeader for given number of tags.....	APR-261
LoadATAG	Load an ATAG simple audio object format file.	APR-262
ValidateAudioTagHeader	Validate the contents of an AudioTagHeader (ATAG chunk).	APR-263

Juggler_Classes

CollectionClass	Multiple parallel sequences and collections.	APR-264
JuggleeClass	Root juggler class.	APR-265
SequenceClass	A single sequence of events.....	APR-266

Juggler_Functions

AbortObject	Abnormally stops a juggler object.	APR-267
AllocObject	Allocates memory for an object.	APR-268
BumpJuggler	Bumps the juggler data-structure index.....	APR-269
CreateObject	Creates an object of the given class.....	APR-271
DefineClass	Defines a class of objects.	APR-272
DestroyObject	Destroys an object.	APR-273
FreeObject	Frees memory allocated for an object.	APR-274
GetNthFromObject	Gets the nth element of a collection.....	APR-275
GetObjectInfo	Gets the current settings of an object.	APR-276
InitJuggler	Initializes the juggler mechanism for controlling events.	APR-277
PrintObject	Prints debugging information about an object.	APR-278
RemoveNthFromObject	Removes the nth element of a collection.	APR-279
SetObjectInfo	Sets values in the object based on tag args.	APR-280
StartObject	Starts an object so the juggler will play it.	APR-282
StopObject	Stops an object so the juggler won't play it.	APR-283
TermJuggler	Terminates the juggler mechanism for controlling events.	APR-284
ValidateObject	Validates an object.....	APR-285

Sample

DeleteSampleInfo.....	Frees SampleInfo.....	APR-286
DumpSampleInfo.....	Dumps SampleInfo structure returned by \f4GetAIFFSampleInfo()\fP.....	APR-287
GetAIFFSampleInfo.....	Parses an AIFF sample file and return a SampleInfo data structure.....	APR-288
LoadSample.....	Loads a \f4Sample\fP from an AIFF or AIFC file.....	APR-290
LoadSystemSample.....	Loads a system \f4Sample\fP file.....	APR-292
SampleFormatToInsName.....	Builds the name of the DSP Instrument Template to play a sample of the specified format.....	APR-294
SampleInfoToTags.....	Fills out a tag list to create a \f4Sample\fP item from SampleInfo.....	APR-296
SampleItemToInsName.....	Builds the name of the DSP Instrument Template to play the \f4Sample\fP.....	APR-298
UnloadSample.....	Unloads a sample loaded by \f4LoadSample()\fP.....	APR-300

Score

ChangeScoreControl.....	Changes a MIDI control value for a channel.....	APR-301
ChangeScorePitchBend.....	Changes a channel's pitch bend value.....	APR-303
ChangeScoreProgram.....	Changes the MIDI program for a channel.....	APR-305
ConvertPitchBend.....	Converts a MIDI pitch bend value into frequency multiplier.....	APR-306
CreateScoreContext.....	Allocates a score context.....	APR-308
CreateScoreMixer.....	Creates and initializes a mixer instrument for MIDI score playback.....	APR-309
DeleteScoreContext.....	Deletes a score context.....	APR-311
DeleteScoreMixer.....	Disposes of mixer created by \f4CreateScoreMixer()\fP.....	APR-312
DisableScoreMessages.....	Enable or disable printed messages during score playback.....	APR-313
FreeChannelInstruments.....	Frees all of a MIDI channel's instruments.....	APR-314
GetScoreBendRange.....	Gets the current pitch bend range value for a score context.....	APR-315
InitScoreDynamics.....	Sets up dynamic voice allocation.....	APR-316
InterpretMIDIEvent.....	Interprets MIDI events within a MIDI object.....	APR-317
InterpretMIDIMessage.....	Executes a MIDI message.....	APR-318
LoadPIMap.....	Loads a Program-Instrument Map (\f4PIMap\fP) from a text file.....	APR-320
LoadScoreTemplate.....	Loads instrument or patch \f4Template\fP depending on file name extension.....	APR-321
MfDefineCollection.....	Creates a juggler collection from a MIDI file image in RAM.....	APR-323
MfLoadCollection.....	Loads a set of sequences from a MIDI file.....	APR-324
MfLoadSequence.....	Loads a sequence from a MIDI file.....	APR-325
MfUnloadCollection.....	Unloads a MIDI collection.....	APR-326
NoteOffIns.....	Turns off a note played by an instrument.....	APR-327
NoteOnIns.....	Turns on a note for an instrument.....	APR-328
PIMap.....	Program-Instrument Map file format.....	APR-329
PurgeScoreInstrument.....	Purges an unused instrument from a ScoreContext.....	APR-331
ReleaseScoreNote.....	Releases a MIDI note, uses voice allocation.....	APR-333
SetPIMapEntry.....	Specifies the instrument to use when a MIDI program change occurs.....	APR-334
SetScoreBendRange.....	Sets the current pitch bend range value for a score context.....	APR-336
StartScoreNote.....	Starts a MIDI note, uses voice allocation.....	APR-337
StopScoreNote.....	Stop a MIDI note immediately with no release phase.....	APR-338
UnloadPIMap.....	Unloads instrument templates loaded previously with PIMap file.....	APR-339
UnloadScoreTemplate.....	Unloads a \f4Template\fP loaded by \f4LoadScoreTemplate()\fP.....	APR-340

Sound3D

Create3DSound	Allocate and initialize the 3D Sound data structures.	APR-341
Delete3DSound	Release resources and deallocate 3D Sound data structures.	APR-345
Get3DSoundInstrument	Returns the DSP instrument to which a source should be connected to use a 3D Sound.	APR-346
Get3DSoundParms	Return application-dependent cue parameters.	APR-347
Get3DSoundPos	Calculate the instantaneous position of a 3D Sound at the time of the call.	APR-349
Move3DSound	Moves a 3D Sound in a line from start to end over a given time period.	APR-350
Move3DSoundTo	Moves a 3D Sound in a line from wherever it currently is to a given endpoint.	APR-352
s3dNormalizeAngle	Normalize an angle to be between -PI and +PI.	APR-353
Start3DSound	Starts a 3D Sound, and positions it in space.	APR-354
Stop3DSound	Stops a 3D Sound.	APR-355

SoundPlayer

spAddMarker	Add a new SPMarker to an SPSound.	APR-356
spAddSample	Create an SPSound for a Vf4SamplefP Item.	APR-357
spAddSoundFile	Create an SPSound for an AIFF sound file.	APR-360
spBranchAtMarker	Set up a static branch at a marker.	APR-363
spClearDefaultDecisionFunction	Clears global decision function.	APR-365
spClearMarkerDecisionFunction	Clears a marker decision function.	APR-366
spContinueAtMarker	Clear static branch at a marker.	APR-367
spCreatePlayer	Create an SPPlayer.	APR-368
SPDecisionFunction	Typedef for decision callback functions.	APR-371
spDeletePlayer	Delete an SPPlayer.	APR-374
spDumpPlayer	Print debug information for sound player.	APR-375
spFindMarkerName	Return pointer an SPMarker by looking up its name.	APR-376
spGetMarkerName	Get name of an SPMarker.	APR-377
spGetMarkerPosition	Get position of an SPMarker.	APR-378
spGetPlayerFromSound	Get SPPlayer that owns an SPSound.	APR-379
spGetPlayerSignalMask	Get set of signals player will send to client.	APR-380
spGetPlayerStatus	Get current status of an SPPlayer.	APR-381
spGetSoundFromMarker	Get SPSound that owns an SPMarker.	APR-382
spIsSoundInUse	Determines if SPPlayer is currently reading from a particular SPSound.	APR-383
spLinkSounds	Branch at end of one sound to beginning of another.	APR-384
spLoopSound	Branch at end of sound back to the beginning.	APR-385
spPause	Pause an SPPlayer.	APR-386
spRemoveMarker	Manually remove an SPMarker from an SPSound.	APR-387
spRemoveSound	Manually remove an SPSound from an SPPlayer.	APR-388
spResume	Resume playback of an SPPlayer after being paused.	APR-390
spService	Service SPPlayer.	APR-391
spSetBranchAction	Set up an SPAction to branch to the specified marker.	APR-393
spSetDefaultDecisionFunction	Install a global decision function to be called for every marker.	APR-394
spSetMarkerDecisionFunction	Install a marker decision function.	APR-395
spSetStopAction	Set up an SPAction to stop reading.	APR-396
spStartPlaying	Begin emitting sound for an SPPlayer.	APR-397

spStartReading	Start SPPlayer reading from an SPSound	APR-399
spStop	Stops an SPPlayer.	APR-401
spStopAtMarker	Stop when playback reaches marker.	APR-402

SoundSpooler

SoundBufferFunc	SoundSpooler callback function typedef	APR-403
ssplAbort	Aborts SoundSpooler buffers in active queue	APR-405
ssplAttachInstrument	Attaches new sample player instrument to SoundSpooler	APR-406
ssplCreateSoundSpooler	Creates a SoundSpooler.	APR-407
ssplDeleteSoundSpooler	Deletes a SoundSpooler	APR-408
ssplDetachInstrument	Detaches the current sample player instrument from the SoundSpooler. .	APR-409
ssplDumpSoundBufferNode ..	Print debug info for SoundBufferNode	APR-410
ssplDumpSoundSpooler	Print debug info for SoundSpooler.	APR-411
ssplGetSBMsgClass	Get class of message passed to \f4SoundBufferFunc\fP.	APR-412
ssplGetSequenceNum	Gets Sequence Number of a SoundBufferNode.	APR-413
ssplGetSpoolerStatus	Gets SoundSpooler status flags	APR-414
ssplGetUserData	Gets User Data from a SoundBufferNode.	APR-415
ssplIsSpoolerActive	Tests if spooler has anything in its active queue	APR-416
ssplPause	Pauses the SoundSpooler	APR-417
ssplPlayData	Waits for next available buffer then sends a block full of data to the spooler (convenience function)	APR-418
ssplProcessSignals	Processes completion signals that have been received	APR-419
ssplRequestBuffer	Asks for an available buffer	APR-421
ssplReset	Resets SoundSpooler.	APR-423
ssplResume	Resume SoundSpooler playback.	APR-424
ssplSendBuffer	Sends a buffer full of data	APR-425
ssplSetBufferAddressLength ..	Attaches sample data to a SoundBufferNode	APR-426
ssplSetSoundBufferFunc	Install new \f4SoundBufferFunc\fP in SoundSpooler.	APR-427
ssplSetUserData	Stores user data in a SoundBufferNode	APR-428
ssplSpoolData	Sends a block full of data. (convenience function)	APR-429
ssplStartSpoolerTags	Starts SoundSpooler	APR-430
ssplStopSpooler	Stops SoundSpooler	APR-431
ssplUnrequestBuffer	Returns an unused buffer to the sound spooler	APR-432
UserBufferProcessor	Callback function prototype called by \f4ssplProcessSignals()\fP, \f4ssplAbort()\fP, and \f4ssplReset()\fP	APR-433

Volume 4:

3DO M2 Portfolio Programmer's Guide

Preface

About This Book	PPG-xxiii
Programmer's Guide vs. Programmer's Reference	PPG-xxiv
How This Book is Organized	PPG-xxiv
Related Documentation	PPG-xxv
About the Code Examples	PPG-xxvi
About Bit Ordering	PPG-xxvii
Typographical Conventions	PPG-xxvii

1

Understanding the Kernel

Description of the Kernel	PPG-1
Multitasking	PPG-2
Multitasking	PPG-2
Task Termination	PPG-4
Parent Tasks, Child Tasks, and Threads	PPG-4
Memory Management	PPG-5
Memory Size	PPG-5
Allocating Memory	PPG-5
Memory Fences	PPG-8
Returning Memory	PPG-9
Sharing Memory	PPG-9
Working with Folios	PPG-10
Managing Items	PPG-10
Items	PPG-10
Creating an Item	PPG-11
Opening Items	PPG-11
Using Items	PPG-11
Deleting Items	PPG-12
Semaphores	PPG-12
Intertask Communication	PPG-13
Signals	PPG-13
Messages	PPG-14
Portfolio Error Codes	PPG-16

2

Tasks and Threads

What Is a Task?	PPG-19
Starting and Ending Tasks	PPG-20
Creating a Task	PPG-20
Ending a Task	PPG-21
Controlling the State of a Task	PPG-22
Yielding the CPU to Another Task	PPG-22

Going To and From Wait State	PPG-22
Changing a Task's Parent	PPG-23
Changing Task Priorities	PPG-23
Task Quantums	PPG-24
Starting and Ending Threads	PPG-24
Creating a Thread	PPG-24
Ending a Thread	PPG-26
Advanced Task and Thread Usage	PPG-26
Example: Using Threads and Signals.....	PPG-27

3

Using Tag Arguments

About Tag Arguments.....	PPG-31
Special Tag Commands	PPG-32
How to Specify Tags.....	PPG-33
How Tags are Documented	PPG-33
How to Specify Tags Using a Tag Array	PPG-33
How to Specify Tags Using VarArg	PPG-34
Parsing Tags.....	PPG-35

4

Managing Linked Lists

About Linked Lists.....	PPG-38
Characteristics of Portfolio Lists	PPG-38
Characteristics of Nodes	PPG-39
Creating and Initializing a List.....	PPG-39
Adding a Node to a List	PPG-40
Adding a Node to the Head of a List	PPG-40
Adding a Node to the Tail of a List	PPG-40
Adding a Node After Another Node in a List	PPG-40
Adding a Node Before Another Node in a List	PPG-41
Adding a Node According to Its Priority	PPG-41
Adding a Node in Alphabetical Order	PPG-41
Adding a Node According to Other Node Values	PPG-42
Changing the Priority of a Node.....	PPG-42
Removing a Node From a List	PPG-42
Removing the First Node	PPG-42
Removing the Last Node	PPG-43
Removing a Specific Node	PPG-43
Finding Out If a List Is Empty	PPG-43
Traversing a List.....	PPG-43
Traversing a List From Front to Back	PPG-43
Traversing a List From Back to Front	PPG-44
Finding a Node by Name.....	PPG-45
Finding a Node by Its Ordinal Position.....	PPG-45
Determining the Ordinal Position of a Node.....	PPG-46
Counting the Number of Nodes in a List.....	PPG-46

Primary Data Structures	PPG-46
The Node Structure	PPG-46
MinNode and NamelessNode Structures	PPG-47
The List Data Structure	PPG-48
The ListAnchor Union	PPG-48
The Link Data Structure	PPG-49

5 Managing Memory

About Memory	PPG-52
How Memory Works	PPG-52
Managing Memory	PPG-53
Allocating a Block of Memory	PPG-53
Freeing a Block of Memory	PPG-54
Reallocating a Block of Memory	PPG-54
Getting Information About Memory	PPG-55
Finding Out How Big a Block of Memory Is	PPG-55
Finding Out How Much Memory Is Available	PPG-55
Getting the Size of a Memory Page	PPG-56
Validating Memory Pointers	PPG-56
Allowing Other Tasks to Write to Your Memory	PPG-57
Transferring Memory to Other Tasks	PPG-58
Interfacing to the System's Free Page List Directly	PPG-58
Debugging Memory Usage	PPG-59
Example: Allocating and Deallocating Memory	PPG-61
Other Memory Topics (Caches, Bit Arrays)	PPG-62
Caches	PPG-62
Bit Arrays	PPG-64

6 Managing Items

About Items	PPG-67
Creating or Opening an Item	PPG-69
Creating an Item	PPG-69
Specifying the Item Type	PPG-69
Tag Arguments	PPG-70
VarArgs Tag Arguments	PPG-71
The Item Number	PPG-72
Convenience Calls to Create Items	PPG-72
Using an Item	PPG-73
Finding an Item	PPG-73
Getting a Pointer to an Item	PPG-73
Checking If an Item Exists	PPG-74
Changing Item Ownership	PPG-74
Changing an Item's Priority	PPG-74

Deleting an Item	PPG-75
Opening an Item	PPG-75
Closing an Item.....	PPG-75

7

Semaphores

About Semaphores	PPG-77
The Problem: Sharing Resources Safely	PPG-78
The Solution: Locking Shared Resources When Using Them	PPG-78
The Implementation: Semaphores	PPG-78
Using Semaphores	PPG-78
Creating a Semaphore	PPG-79
Locking a Semaphore	PPG-79
Exclusive and Shared Access.....	PPG-80
Priority Inversion	PPG-80
Unlocking a Semaphore.....	PPG-80
Finding a Semaphore	PPG-81
Deleting a Semaphore.....	PPG-81

8

Communicating Among Tasks

About Intertask Communication	PPG-83
Using Signals.....	PPG-84
Allocating a Signal Bit	PPG-84
Receiving a Signal	PPG-85
Sampling and Changing the Current Signal Bits	PPG-86
Sending a Signal	PPG-86
Freeing Signal Bits	PPG-86
Sample Code for Signals	PPG-87
Passing Messages	PPG-87
Creating a Message Port	PPG-88
Creating a Message	PPG-89
Sending a Message	PPG-91
Receiving a Message	PPG-92
Working with a Message	PPG-93
Pulling Back a Message	PPG-94
Replying to a Message	PPG-94
Messages With No Reply Ports	PPG-95
Forwarding Messages to Another Port	PPG-96
Finding a Message Port	PPG-96
Example Code for Messages	PPG-96

9

The Portfolio I/O Model

Introduction	PPG-101
Hardware Devices and Connections.....	PPG-102
The Control Port	PPG-102

I/O Architecture	PPG-103
Devices	PPG-103
Device Stacks	PPG-104
Drivers	PPG-104
I/O Requests (IOReqs)	PPG-104
Performing I/O.....	PPG-105
Preparing for I/O	PPG-105
Creating an IOReq	PPG-106
Initializing an IOInfo Structure	PPG-107
Passing IOInfo Values to the Device	PPG-109
Asynchronous and Synchronous I/O	PPG-109
Completion Notification	PPG-111
Reading an IOReq	PPG-112
Continuing I/O	PPG-113
Aborting I/O	PPG-113
Finishing I/O	PPG-114

10 Portfolio Devices

Introduction.....	PPG-115
Timer Devices.....	PPG-117
Creating a Timer IOReq	PPG-117
The Microsecond Clock	PPG-117
The Vertical Blank Clock	PPG-120
File Devices.....	PPG-123
Important Note About Microcard File Systems	PPG-123
Methods for Communicating With File Devices	PPG-123
Creating, Deleting, Renaming, Getting Information about Files and Directories	PPG-124
Preparing to Send an IOReq to a File Device	PPG-124
Getting Filesystem Status	PPG-125
Getting File Status	PPG-126
Allocating Blocks for a File	PPG-127
Writing Blocks of Data to a File	PPG-127
Marking the End of a File	PPG-128
Reading Blocks of Data From a File	PPG-128
Setting the File Type	PPG-129
Setting the File Version and Revision	PPG-129
Setting the Virtual Block Size	PPG-129
Getting Directory Information	PPG-130
Cleaning Up After I/O Operations	PPG-131
Serial Devices.....	PPG-131
Writing Data to a Serial Port	PPG-131
Reading Data from a Serial Port	PPG-132
Configuring the Serial Port	PPG-133
Reading the Serial Port Configuration	PPG-134
Waiting for Particular Events	PPG-134

Controlling Serial Lines	PPG-135
Sending Break Signals	PPG-135
Getting Serial State	PPG-135

11

The File System, File Folio, and FSUtils Folio

The 3DO File System Interface.....	PPG-137
Files and File Functions	PPG-138
RawFiles and RawFile Functions	PPG-138
Directories and Directory Functions	PPG-138
Avatars	PPG-139
Pathnames and Filenames	PPG-140
File Functions.....	PPG-141
Creating and Deleting Files	PPG-141
Renaming Files and Directories	PPG-141
Opening and Closing Files (For Block-Oriented Operations)	PPG-141
Setting File Attributes	PPG-142
Finding Files	PPG-142
RawFile Functions.....	PPG-145
Opening a RawFile	PPG-146
Reading From a RawFile	PPG-147
Writing to a RawFile	PPG-147
Setting the Size of a RawFile	PPG-148
Seeking in a RawFile	PPG-148
Clearing Error Conditions for an Open RawFile	PPG-149
Getting Information About a RawFile	PPG-149
Setting the Attributes of a RawFile	PPG-150
Closing a RawFile	PPG-151
Directory Functions	PPG-151
Creating and Deleting Directories	PPG-151
Opening and Closing Directories	PPG-151
Finding and Changing the Current Directory	PPG-152
Reading a Directory	PPG-153
Mounting and Dismounting File Systems	PPG-153
Minimizing File Systems	PPG-153
Finding Mounted File Systems	PPG-154
File Folio Examples	PPG-154
Displaying Contents of a File	PPG-155
Scanning the File System	PPG-159
Listing the Contents of a Directory	PPG-162
The FSUtils Folio - File System Utilities.....	PPG-165
Copying a Directory Tree	PPG-165
Deleting a Directory Tree	PPG-168
Determining the Full Pathname of an Open File	PPG-169
Extracting the Final Component of a Pathname	PPG-169
Appending a Pathname to Another Pathname	PPG-169

12

The Batt Folio and Date Folio

Introduction - The Battery-Backed Clock	PPG-171
The Batt Folio	PPG-172
Reading the Current Gregorian Date and Time	PPG-172
Setting the Current Gregorian Date and Time	PPG-172
The Date Folio	PPG-173
Converting from Gregorian to TimeVal Representation	PPG-173
Converting from TimeVal to Gregorian Representation	PPG-173
Validating a Gregorian Date-Time Value	PPG-174

13

The Event Broker

About the Event Broker	PPG-176
Specifying and Monitoring Events	PPG-176
Working With Input Focus	PPG-177
Reconfiguring or Disconnecting an Event Broker Connection	PPG-178
Other Event Broker Activities	PPG-178
Sending Messages Between Tasks and the Event Broker	PPG-178
Message Flavors	PPG-178
Message Types	PPG-181
Flavor-Specific Message Requirements	PPG-182
The Process of Event Monitoring	PPG-182
Connecting a Task to the Event Broker	PPG-183
Creating a Message Port	PPG-183
Creating a Configuration Message	PPG-183
Creating a Configuration Block	PPG-183
Sending the Configuration Message	PPG-189
Receiving the Configuration Reply Message	PPG-189
Monitoring Events Through the Event Broker	PPG-190
Waiting for Event Messages on the Reply Port	PPG-190
Reading an Event Message Data Block	PPG-191
Reading Event Data	PPG-194
High-Performance Event Broker Use	PPG-199
The Pod Table	PPG-199
Gaining Access to the Pod Table	PPG-200
Relinquishing Access to the Pod Table	PPG-200
Structure of the Pod Table	PPG-200
Empty Generic Class Substructures	PPG-202
Non-Empty Generic Class Substructures	PPG-202
Use and Abuse of the Pod Table	PPG-204
Synchronization Between the Pod Table and Event Messages	PPG-206
Reconfiguring or Disconnecting a Task	PPG-206
Reconfiguring the Event Broker Connection	PPG-206
Disconnecting From the Event Broker	PPG-207
Other Event Broker Activities	PPG-207

Getting Lists of Listeners and Connected Pods	PPG-207
Working With Input Focus	PPG-210
Commanding a Pod	PPG-211
Event Broker Convenience Calls.....	PPG-214
Connecting to the Event Broker	PPG-214
Monitoring a Control Pad or a Mouse	PPG-215
Disconnecting From the Event Broker	PPG-216

14

The International Folio

What Is Internationalization?	PPG-217
The Locale Data Structure	PPG-218
NumericSpec Structure	PPG-220
DateSpec Arrays	PPG-223
Using the Locale Structure	PPG-223
Determining the Current Language and Country	PPG-223
Working With International Character Strings	PPG-224
Formatting Numbers or Currency	PPG-226
Formatting Dates	PPG-226

15

The Compression Folio

Introduction.....	PPG-227
How the Compression Folio Works.....	PPG-228
Compressing Data	PPG-228
Decompressing Data	PPG-228
How to Use the Compression Folio.....	PPG-228
The Callback Function.....	PPG-230
Controlling Memory Allocations	PPG-230
Convenience Calls.....	PPG-231
Example: Using Compression.....	PPG-231

16

The IFF Folio

Overview of the IFF File Format and Folio	PPG-237
The EA IFF 85 Standard	PPG-237
The IFF File Format	PPG-238
Goals of the IFF Format	PPG-239
The Purpose and Capabilities of the IFF Folio	PPG-240
How IFF Formats and Folio are Used in 3DO M2 System	PPG-243
Description of the IFF File Format.....	PPG-243
Standard Chunk Format	PPG-243
3DO Extension to Standard Chunk Format	PPG-245
Container Chunks	PPG-246
Using the IFF Folio - a Tutorial.....	PPG-250
Reading and Processing a FORM	PPG-250
Writing out a FORM	PPG-254

Description of IFF Folio Functions	PPG-255
Setting Up, Starting, and Terminating a Parse Operation	PPG-256
Defining, Saving, and Finding Collection Chunks	PPG-257
Defining, Saving, and Finding Property Chunks	PPG-259
Defining Stop Chunks	PPG-262
Defining Entry and Exit Handlers	PPG-263
Using ContextInfo Structures	PPG-264
Finding Context Nodes	PPG-268
Pushing and Popping Context Nodes	PPG-268
Reading and Writing Chunks	PPG-269
Positioning the Cursor in a Stream	PPG-270

17

The Icon Folio

Introduction - What the Icon Folio Does	PPG-271
About Icons	PPG-271
Functions Provided by the Icon Folio	PPG-272
Loading an Icon into Memory.....	PPG-272
Unloading an Icon from Memory.....	PPG-274
Saving an Icon.....	PPG-274

18

The SaveGame Folio

Introduction - What the SaveGame Folio Does	PPG-275
What Kind of Data is Saved - Determined by Application	PPG-275
Functions Provided by the SaveGame Folio	PPG-276
Saving Game Data	PPG-276
Loading Saved Game Data into Memory.....	PPG-278

19

The Requestor Folio

Introduction - What the Requestor Folio Does.....	PPG-281
Purpose	PPG-281
How to Use the Requestor Folio - Overview	PPG-282
Creating a Requestor Object.....	PPG-283
Displaying the Storage Requestor Interface to the User	PPG-285
Querying a Storage Requestor Object.....	PPG-285
Modifying a Storage Requestor Object	PPG-286
Deleting a Storage Requestor Object.....	PPG-286

20

Dynamic-Link Libraries

About DLLs.....	PPG-287
Creating and Using 3DO M2 DLLs	PPG-288
Modules	PPG-288
Linking Multi-Module Applications	PPG-289
Definition Files.....	PPG-290

How Definition Files Work	PPG-290
Example of a Definition File	PPG-291
Building a Multi-Module Application	PPG-291
Files Used in a Multi-Module Application	PPG-292
Building an Exporting Module	PPG-294
Building and Executing an Importing Module	PPG-295
Linking an Application with a DLL	PPG-295
Functions for Handling Modules	PPG-296
Using Items as Arguments and Return Values	PPG-296
Opening, Loading, and Executing Modules	PPG-296
Importing and Exporting Modules	PPG-297
Removing Symbols from the List of Available Symbols	PPG-299
Looking Up the Address of a Symbol	PPG-299
Example: Creating and Using a DLL	PPG-300
Building a DLL Step by Step	PPG-300
Importing a DLL Step by Step	PPG-301
Executing the IMPORTER Program	PPG-301
Output of the IMPORTER Program	PPG-301

21

The Debug Console Link Library

About the Debug Console Link Library	PPG-311
Preparing For Console Output	PPG-312
Performing Console Output	PPG-312
Concluding Console Output	PPG-313

22

The Script Folio

About the Script Folio	PPG-315
Executing Commands	PPG-316
Preserving State Across Multiple Executions	PPG-316

23

Using Lumberjack, the Event Logger

Overview	PPG-319
Collecting Information With Lumberjack	PPG-320
Creating Lumberjack	PPG-320
Starting and Stopping Event Logging	PPG-320
Logging Custom Events	PPG-320
Deleting Lumberjack	PPG-321
Getting Information From Lumberjack	PPG-321
Getting and Releasing Buffers	PPG-321
Parsing Lumberjack Buffers	PPG-322
Example: Using Lumberjack	PPG-323

Index	PPG-325
-------------	---------

3DO M2 Portfolio Programmer's Reference

1

ANSI C Link Library Calls

printf.....	Standard C formatting routines.....	PPR-3—7
fprintf.....	Standard C formatting routines.....	PPR-3—7
sprintf.....	Standard C formatting routines.....	PPR-3—7
vprintf.....	Standard C formatting routines.....	PPR-3—7
vfprintf.....	Standard C formatting routines.....	PPR-3—7
vsprintf.....	Standard C formatting routines.....	PPR-3—7
cprintf.....	Standard C formatting routines.....	PPR-3—7
vcprintf.....	Standard C formatting routines.....	PPR-3—7

Math

abs.....	Absolute Value.....	PPR-8
labs.....	Absolute Value.....	PPR-8

2

Batt Folio Calls

ReadBattClock.....	Reads the current setting of the battery-backed clock.....	PPR-11
WriteBattClock.....	Sets the battery-backed clock.....	PPR-12

3

Compression Folio Calls

CreateCompressor.....	Creates a compression engine.....	PPR-15
CreateDecompressor.....	Creates a decompression engine.....	PPR-17
DeleteCompressor.....	Deletes a compression engine.....	PPR-19
DeleteDecompressor.....	Deletes a decompression engine.....	PPR-20
FeedCompressor.....	Gives data to a compression engine.....	PPR-21
FeedDecompressor.....	Gives data to a decompression engine.....	PPR-22
GetCompressorWorkBufferSize.....	Gets the size of the work buffer needed by a compression engine.....	PPR-23
GetDecompressorWorkBufferSize.....	Gets the size of the work buffer needed by a decompression engine.....	PPR-24
SimpleCompress.....	Compresses some data in memory.....	PPR-25
SimpleDecompress.....	Decompresses some data in memory.....	PPR-26

4

Date Folio Calls

ConvertGregorianToTimeVal.....	Converts from a real-world date representation to a system TimeVal.....	PPR-29
ConvertTimeValToGregorian.....	Converts from a system TimeVal to a real-world date representation.....	PPR-30
ValidateDate.....	Makes sure a date is valid.....	PPR-31

5

Debug Console Link Library Calls

CreateDebugConsole.....	Creates a view for debugging console output.....	PPR-35
DebugConsoleClear.....	Erases everything in a debugging view console.....	PPR-36
DebugConsoleMove.....	Moves the debugging console rendering cursor.....	PPR-37

DebugConsolePrintf.....	Output information to a debugging view console.....	PPR-38
DeleteDebugConsole.....	Closes the debugging console view preventing further debugging output..	PPR-39

6

Device Command Reference

CD-ROM

CDROMCMD_CLOSE_DRAWER	Close the CD-ROM drive's drawer.	PPR-43
CDROMCMD_DISCDATA	Requests the Disc Information (DiscID, TOC, and Session Info).	PPR-44
CDROMCMD_OPEN_DRAWER	Open the CD-ROM drive's drawer.....	PPR-45
CDROMCMD_READ	Requests specific block (sector) data from the disc.	PPR-46
CDROMCMD_READ_SUBQ	Requests the QCode data for the current sector.	PPR-50
CDROMCMD_SCAN_READ.....	Requests approximate block (sector) data from the disc.....	PPR-51
CDROMCMD_SETDEFAULTS..	Set/Get the current device defaults for read-mode IOReqs.	PPR-55

File

FILECMD_ALLOCBLOCKS	Allocate storage space for a file.	PPR-59
FILECMD_FSSTAT	Get filesystem status information.	PPR-60
FILECMD_GETPATH	Get pathname of a file.....	PPR-61
FILECMD_READDIR.....	Read a directory entry by number.....	PPR-62
FILECMD_READENTRY	Read a directory entry by name.	PPR-63
FILECMD_SETEOF	Set the End Of File (EOF).....	PPR-64
FILECMD_SETTYPE.....	Set file type.....	PPR-65
FILECMD_SETVERSION.....	Set revision and version of a file.....	PPR-66

Generic

CMD_BLOCKREAD	Reads data from a block-oriented device.....	PPR-67
CMD_BLOCKWRITE.....	Writes data to a block-oriented device.....	PPR-68
CMD_GETMAPINFO.....	Returns information about the memory-mapping capabilities of a device. .	PPR-69
CMD_MAPRANGE	Requests a device to map itself into memory.	PPR-70
CMD_PREFER_FSTYPE	Requests the preferred filesystem type for a device.....	PPR-71
CMD_STATUS	Requests the DeviceStatus information for a device.....	PPR-72
CMD_STREAMREAD	Reads data from a stream-oriented device.....	PPR-73
CMD_STREAMWRITE.....	Writes data to a stream-oriented device.....	PPR-74
CMD_UNMAPRANGE.....	Requests a device to unmap itself from memory.....	PPR-75

Graphics

GFXCMD_EXECUTETECMDS ..	Submits a list of rendering commands for execution by the Triangle Engine.	PPR-76
-------------------------	--	--------

Host

HOST_CMD_RECV.....	Waits for a packet of information from a remote host development system.....	PPR-78
HOST_CMD_SEND	Sends a packet of information to a remote host development system.....	PPR-79

HostConsole

HOSTCONSOLE_CMD_GETCMDLINE	Gets command-line input from the remote host.	PPR-80
----------------------------	--	--------

HostFS

HOSTFS_CMD_ALLOCBLOCKS	Controls the number of blocks allocated to a file on a remote host file system.	PPR-81
HOSTFS_CMD_BLOCKREAD ..	Reads data from an opened file on a remote host file system.....	PPR-82
HOSTFS_CMD_BLOCKWRITE	Writes data to a remote host file system.....	PPR-83
HOSTFS_CMD_CLOSEENTRY	Concludes use of a reference token.	PPR-84
HOSTFS_CMD_CREATEDIR	Creates a new directory within an existing directory.....	PPR-85
HOSTFS_CMD_CREATEFILE	Creates a new file within an existing directory.	PPR-86
HOSTFS_CMD_DELETEENTRY	Deletes an entry within a remote file system.....	PPR-87
HOSTFS_CMD_DISMOUNTFS	Requests that a file system be dismounted.	PPR-88
HOSTFS_CMD_FSSTAT	Obtains information about a remote host file system.	PPR-89
HOSTFS_CMD_MOUNTFS	Requests that a file system be mounted on a remote host.	PPR-90
HOSTFS_CMD_OPENENTRY ..	Obtains a reference token for an object within a remote file system.	PPR-91
HOSTFS_CMD_READDIR	Obtains information about an object within a remote host file system.....	PPR-92
HOSTFS_CMD_READENTRY	Obtains information about an object within a remote host file system.....	PPR-93
HOSTFS_CMD_RENAMEENTRY	Renames an object on a remote host file system.....	PPR-94
HOSTFS_CMD_SETBLOCKSIZE	Sets the device block size for a particular IOReq dealing with a file on a remote host file system.....	PPR-95
HOSTFS_CMD_SETEOF	Sets the logical byte count of a file on a remote host file system.	PPR-96
HOSTFS_CMD_SETTYPE.....	Sets the four byte file type for an object on a remote host file system.....	PPR-97
HOSTFS_CMD_SETVERSION	Set the version and revision codes for an opened entry on a remote host file system.	PPR-98
HOSTFS_CMD_STATUS	Obtains information about an opened entry on a remote host file system. .	PPR-99

MPEG

MPEGVIDEOCMD_CONTROL .	Permits application control over MPEG video decoding.	PPR-100
MPEGVIDEOCMD_READ	Submits a bitmap item for the MPEG video device to place a decoded picture.	PPR-102
MPEGVIDEOCMD_WRITE.....	Submits a buffer of MPEG video data for decoding by the MPEG video device.	PPR-103

Serial

SER_CMD_BREAK.....	Sends a break signal over the serial line.....	PPR-104
SER_CMD_GETCONFIG.....	Determines the current serial port settings.....	PPR-105
SER_CMD_SETCONFIG.....	Sets the configuration of the serial port.....	PPR-106
SER_CMD_SETDTR.....	Controls the state of the serial DTR line.....	PPR-107
SER_CMD_SETLOOPBACK.....	Controls the state of automatic serial loopback mode.....	PPR-108
SER_CMD_SETRTS.....	Controls the state of the serial RTS line.....	PPR-109
SER_CMD_STATUS.....	Gets the state of various serial attributes.....	PPR-110
SER_CMD_WAITEVENT.....	Waits for serial events to occur.....	PPR-111

Timer

TIMERCMD_DELAYUNTIL_USEC.....	Waits for a given time.....	PPR-112
TIMERCMD_DELAYUNTIL_VBL.....	Waits for the system VBL count to reach a specific number.....	PPR-113
TIMERCMD_DELAY_USEC.....	Waits for a fixed amount of time to pass.....	PPR-114
TIMERCMD_DELAY_VBL.....	Waits for a fixed number of vertical blanking intervals.....	PPR-115
TIMERCMD_GETTIME_USEC.....	Returns the current system time.....	PPR-116
TIMERCMD_GETTIME_VBL.....	Returns the current system time in vertical blanking intervals.....	PPR-117
TIMERCMD_METRONOME_USEC.....	Requests to be signalled at regular intervals of time.....	PPR-118
TIMERCMD_METRONOME_VBL.....	Requests to be signalled at regular intervals of time.....	PPR-119
TIMERCMD_SETTIME_USEC.....	Sets the current system time.....	PPR-120
TIMERCMD_SETTIME_VBL.....	Sets the current system time in vertical blanking intervals.....	PPR-121

7

Event Broker Calls

GetControlPad.....	Gets control pad events.....	PPR-125
GetMouse.....	Gets mouse events.....	PPR-126
InitEventUtility.....	Connects task to the event broker.....	PPR-127
KillEventUtility.....	Disconnects a task from the event broker.....	PPR-128

8

File Folio Calls

Directory

ChangeDirectory.....	Changes the current directory.....	PPR-131
ChangeDirectoryInDir.....	Changes the current directory relative to another directory.....	PPR-132
CloseDirectory.....	Closes a directory.....	PPR-133
CreateDirectory.....	Creates a directory.....	PPR-134
CreateDirectoryInDir.....	Creates a directory relative to another directory.....	PPR-135
DeleteDirectory.....	Deletes a directory.....	PPR-136
DeleteDirectoryInDir.....	Deletes a directory relative to another directory.....	PPR-137

GetDirectory	Gets the item number and pathname for the current directory.	PPR-138
OpenDirectoryItem	Opens a directory specified by an item.	PPR-139
OpenDirectoryPath	Opens a directory specified by a pathname.	PPR-140
ReadDirectory	Reads the next entry from a directory.	PPR-141

File

CloseFile	Closes a file.	PPR-143
CreateAlias	Creates a file system alias.	PPR-144
CreateFile	Creates a file.	PPR-145
CreateFileInDir	Creates a file relative to a directory.	PPR-146
DeleteFile	Deletes a file.	PPR-147
DeleteFileInDir	Deletes a file relative to a directory.	PPR-148
FindFileAndIdentify	Searches one or more locations for a file, and returns its pathname.	PPR-149
FindFileAndOpen	Searches one or more locations for a file, and opens the file.	PPR-150
OpenFile	Opens a disk file.	PPR-151
OpenFileInDir	Opens a disk file relative to a directory.	PPR-152
Rename	Rename a file, directory, or filesystem.	PPR-153
SetFileAttrs	Sets some attributes of a file.	PPR-154

RawFile

ClearRawFileError	Clears the error state of a file.	PPR-155
CloseRawFile	Concludes access to a file.	PPR-156
GetRawFileInfo	Gets some information about an opened file.	PPR-157
OpenRawFile	Gains access to a file for raw file I/O.	PPR-158
OpenRawFileInDir()	Gains access to a file for raw file I/O.	PPR-160
ReadRawFile	Reads data from a file.	PPR-162
SeekRawFile	Moves the file cursor within a file.	PPR-164
SetRawFileAttrs	Sets some attributes of an opened file.	PPR-166
SetRawFileSize	Sets the size of an opened file.	PPR-168
WriteRawFile	Writes data to a file.	PPR-169

9

FileSystem Utilities Folio Calls

AppendPath	Appends a path to an existing path specification.	PPR-173
CreateCopyObj	Creates a copier object to perform hierarchical copy operations.	PPR-174
DeleteCopyObj	Releases any resources consumed by \f4CreateCopyObj()\fP.	PPR-177
DeleteTree	Deletes a complete filesystem directory tree.	PPR-178
FindFinalComponent	Returns the last component of a path.	PPR-180
GetPath	Obtain the absolute directory path leading to an opened file.	PPR-181
PerformCopy	Enter the copier engine and start copying files.	PPR-182

10

Icon Folio Calls

LoadIcon	Loads icon data into memory and prepares it for rendering.	PPR-185
Savelcon	Creates an IFF Icon file.	PPR-187
UnloadIcon	Unloads an icon from memory and frees any resources associated with it.	PPR-189

11

IFF Folio Calls

AllocContextInfo	Allocates and initializes a ContextInfo structure.....	PPR-193
AttachContextInfo	Attaches a ContextInfo structure to a given ContextNode.....	PPR-194
CreateIFFParser	Creates a new parser structure and prepares it for use.....	PPR-195
DeleteIFFParser	Deletes an IFF parser.....	PPR-197
FindCollection	Gets a pointer to the current list of collection chunks.....	PPR-198
FindContextInfo	Returns a ContextInfo structure from the context stack.....	PPR-199
FindPropChunk	Searches for a stored property chunk.....	PPR-200
FindPropContext	Gets the parser's current property context.....	PPR-201
FreeContextInfo	Deallocate a ContextInfo structure.....	PPR-202
GetCurrentContext	Gets a pointer to the ContextNode for the current chunk.....	PPR-203
GetIFFOffset	Returns the absolute seek position within the current IFF stream.....	PPR-204
GetParentContext	Gets the nesting ContextNode for the given ContextNode.....	PPR-205
InstallEntryHandler	Adds an entry handler to the parser.....	PPR-207
InstallExitHandler	Adds an exit handler to the parser.....	PPR-208
ParseIFF	Parses an IFF stream.....	PPR-210
PopChunk	Pops the top context node off the context stack.....	PPR-212
PushChunk	Pushes a new context node on the context stack.....	PPR-213
ReadChunk	Reads bytes from the current chunk into a buffer.....	PPR-214
ReadChunkCompressed	Reads and decompresses bytes from the current chunk into a buffer.....	PPR-215
RegisterCollectionChunks	Defines chunks to be collected during the parse operation.....	PPR-216
RegisterPropChunks	Defines property chunks to be stored during the parse operation.....	PPR-217
RegisterStopChunks	Defines chunks that cause the parser to stop and return to the caller.....	PPR-218
RemoveContextInfo	Removes a ContextInfo structure from wherever it is attached.....	PPR-219
SeekChunk	Moves the current position cursor within the current chunk.....	PPR-220
StoreContextInfo	Inserts a ContextInfo structure into the context stack.....	PPR-222
WriteChunk	Writes data from a buffer into the current chunk.....	PPR-223
WriteChunkCompressed	Compressed data from a buffer and writes the result to the current chunk.....	PPR-224

12

International Folio Calls

intlCloseLocale	Terminates use of a given Locale item.....	PPR-227
intlCompareStrings	Compares two strings for collation purposes.....	PPR-228
intlConvertString	Changes certain attributes of a string.....	PPR-229
intlFormatDate	Formats a date in a localized manner.....	PPR-231
intlFormatNumber	Format a number in a localized manner.....	PPR-233
intlGetCharAttrs	Returns attributes describing a given character.....	PPR-236
intlLookupLocale	Returns a pointer to a Locale structure.....	PPR-238
intlOpenLocale	Gains access to a Locale item.....	PPR-239
intlTransliterateString	Converts a string between character sets.....	PPR-240

13

JString Folio Calls

ConvertASCII2ShiftJIS	Converts an ASCII string to Shift-JIS.....	PPR-245
ConvertFullKana2HalfKana.....	Convert a full-width kana string to a half-width kana string.....	PPR-246
ConvertFullKana2Hiragana.....	Convert a full-width kana string to a hiragana string.	PPR-247
ConvertFullKana2Romaji.....	Convert a full-width kana string to a romaji string.	PPR-248
ConvertHalfKana2FullKana.....	Convert a half-width kana string to a full-width kana string.....	PPR-249
ConvertHalfKana2Hiragana	Convert a half-width kana string to a hiragana string.....	PPR-250
ConvertHalfKana2Romaji	Convert a half-width kana string to a romaji string.	PPR-251
ConvertHiragana2FullKana.....	Convert a hiragana string to a full-width kana string.	PPR-252
ConvertHiragana2HalfKana	Convert a hiragana string to a half-width kana string.....	PPR-253
ConvertHiragana2Romaji	Convert a hiragana string to a romaji string.....	PPR-254
ConvertRomaji2FullKana.....	Convert a romaji string to a full-width kana string.	PPR-255
ConvertRomaji2HalfKana	Convert a romaji string to a half-width kana string.	PPR-256
ConvertRomaji2Hiragana	Convert a romaji string to a hiragana string.....	PPR-257
ConvertShiftJIS2ASCII.....	Converts a Shift-JIS string to ASCII.....	PPR-258
ConvertShiftJIS2UniCode	Converts a Shift-JIS string to UniCode.	PPR-259
ConvertUniCode2ShiftJIS	Converts a UniCode string to Shift-JIS.	PPR-260

14

Kernel Folio Calls

BitArrays

AtomicClearBits	Clears bit in a word of memory in an atomic manner.	PPR-263
AtomicSetBits	Sets bit in a word of memory in an atomic manner.	PPR-264
ClearBitRange	Clear a range of bits within a bit array.	PPR-265
CountBits	Count the number of bits set in a word.....	PPR-266
DumpBitRange	Display a range of bits to the debugging terminal.....	PPR-267
FindClearBitRange	Find a range of clear bits within a bit array.	PPR-268
FindLSB	Finds the least-significant bit.	PPR-269
FindMSB	Finds the highest-numbered bit.	PPR-270
FindSetBitRange	Find a range of set bits within a bit array.	PPR-271
IsBitClear	Test whether a bit within a bit array is set to 0.	PPR-272
IsBitRangeClear	Test whether all bits within a range of a bit array are all set to 0.	PPR-273
IsBitRangeSet	Test whether all bits within a range of a bit array are all set to 1.	PPR-274
IsBitSet	Test whether a bit within a bit array is set to 1.	PPR-275
SetBitRange	Set a range of bits within a bit array.	PPR-276

Caches

FlushDCache	Write back modified contents of the data cache to memory, and remove the data from the cache.....	PPR-277
FlushDCacheAll	Write back modified contents of the data cache to memory, and remove the data from the cache.....	PPR-278
GetCacheInfo	Gets information about the CPU caches.	PPR-279
GetDCacheFlushCount	Obtain the current system cache flush count.....	PPR-280
WriteBackDCache	Write back modified contents of the data cache to memory.	PPR-281

DDF

ScanForOOFToken	Scan for a particular token in a token sequence.....	PPR-282
-----------------------	--	---------

Debugging

ControlIOOdebug	Controls what IOOdebug does and doesn't do.....	PPR-283
OdebugBreakpoint	Trigger a breakpoint-like event in the debugger.....	PPR-284
OdebugPutChar	Output a character to the debugging terminal.....	PPR-285
OdebugPutStr	Output a string to the debugging terminal.....	PPR-286
GetSysErr.....	Gets the error string for an error.....	PPR-287
PrintfSysErr	Prints the error string for an error.....	PPR-288

Devices

CloseOdeviceStack.....	Closes a stack of devices.....	PPR-289
CreateOdeviceStackList	Get a list of device stacks which support the requested capabilities.....	PPR-290
OdeleteOdeviceStackList.....	Odestroy the List created by \f4CreateDeviceStackList()\fP.....	PPR-292
OdeviceStackIsIdentical	Compare a DeviceStack to an open device stack.....	PPR-293
OpenOdeviceStack.....	Opens a stack of devices.....	PPR-294

IO

AbortIO	Aborts an I/O operation.....	PPR-295
CheckIO	Checks whether an I/O operation is complete.....	PPR-296
CreateIOReq.....	Creates an I/O request.....	PPR-297
OdeleteIOReq.....	Deletes an I/O request.....	PPR-299
DoIO	Performs synchronous I/O.....	PPR-300
SendIO.....	Requests I/O be performed.....	PPR-301
WaitIO.....	Waits for an I/O operation to complete.....	PPR-303

Items

CheckItem.....	Checks to see if an item exists.....	PPR-304
ClosetItem.....	Closes a previously opened item.....	PPR-305
CreateItem	Creates an item.....	PPR-306
DeleteItem	Odeletes an item.....	PPR-307
FindAndOpenItem	Finds an item by type and tags and opens it.....	PPR-308
FindAndOpenNamedItem	Finds an item by name and opens it.....	PPR-309
FindItem.....	Finds an item by type and tags.....	PPR-310
FindNamedItem	Finds an item by name.....	PPR-311
IsItemOpened	Oetermines whether a task or thread has opened a given item.....	PPR-312
LookupItem	Gets a pointer to an item.....	PPR-313
MkNodeIO.....	Assembles an item type value.....	PPR-314
OpenItem.....	Opens an item.....	PPR-315
SetItemOwner	Changes the owner of an item.....	PPR-316
SetItemPri.....	Changes the priority of an item.....	PPR-317

Lists

AddHead	Adds a node to the head of a list.....	PPR-318
AddTail.....	Adds a node to the tail of a list.....	PPR-319
OumpNode.....	Prints contents of a node.....	PPR-320

FindNamedNode	Finds a node by name.....	PPR-321
FindNodeFromHead	Returns a pointer to a node appearing at a given ordinal position from the head of the list.....	PPR-322
FindNodeFromTail.....	Returns a pointer to a node appearing at a given ordinal position from the tail of the list.....	PPR-323
FirstNode	Gets the first node in a list.....	PPR-324
GetNodeCount	Counts the number of nodes in a list.....	PPR-325
GetNodePosFromHead	Gets the ordinal position of a node within a list, counting from the head of the list.....	PPR-326
GetNodePosFromTail.....	Gets the ordinal position of a node within a list, counting from the tail of the list.....	PPR-327
InsertNodeAfter	Inserts a node into a list after another node already in the list.....	PPR-328
InsertNodeAlpha	Inserts a node into a list according to alphabetical order.....	PPR-329
InsertNodeBefore.....	Inserts a node into a list before another node already in the list.....	PPR-330
InsertNodeFromHead.....	Inserts a node into a list.....	PPR-331
InsertNodeFromTail	Inserts a node into a list.....	PPR-332
IsEmptyList.....	Checks whether a list is empty.....	PPR-333
IsListEmpty.....	Checks whether a list is empty.....	PPR-334
IsNode	Validates a node when moving forward through a list.....	PPR-335
IsNodeB	Validates a node when moving backward through a list.....	PPR-336
LastNode	Gets the last node in a list.....	PPR-337
NextNode	Gets the next node in a list.....	PPR-338
PrepList	Initializes a list.....	PPR-339
PrevNode.....	Gets the previous node in a list.....	PPR-340
RemHead	Removes the first node from a list.....	PPR-341
RemNode	Removes a node from a list.....	PPR-342
RemTail	Removes the last node from a list.....	PPR-343
ScanList.....	Walks through all the nodes in a list.....	PPR-344
ScanListB	Walks through all the nodes in a list backwards.....	PPR-345
SetNodePri	Changes the priority of a list node.....	PPR-346
UniversalInsertNode	Inserts a node into a list.....	PPR-347

Loader

CloseModule	Concludes use of a module item.....	PPR-348
ExecuteModule	Executes code in a module item as a subroutine.....	PPR-349
ExpungeByAddress	Removes a symbol from the list of exports.....	PPR-350
ExpungeBySymbol.....	Removes a symbol from the list of exports.....	PPR-351
ImportByAddress.....	Loads an exporting module based on an imported symbol.....	PPR-352
ImportByName	Loads and resolves the named exporting module.....	PPR-353
LookupSymbol	Returns the address of a loaded symbol.....	PPR-354
OpenModule	Opens an executable file from disk and prepares it for use.....	PPR-355
UnimportByAddress	Unloads an exporting module, based on the address of an import.....	PPR-356
UnimportByName	Unloads a named module.....	PPR-357

Lumberjack

ControlLumberjack	Determine which event types Lumberjack logs.	PPR-358
CreateLumberjack	Initializes Lumberjack, the Portfolio logging service.	PPR-359
DeleteLumberjack	Disables Lumberjack, the Portfolio logging service.	PPR-360
DumpLumberjackBuffer	Parse and display the contents of a Lumberjack buffer.	PPR-361
LogEvent	Add a custom event to the Lumberjack logs.	PPR-362
ObtainLumberjackBuffer	Obtain a pointer to a Lumberjack buffer which is currently full.	PPR-363
ReleaseLumberjackBuffer	Return a logging buffer to the Lumberjack system so it can be reused. ...	PPR-364

Memory

AllocMem	Allocates a block of memory.	PPR-365
AllocMemAligned	Allocates a block of memory aligned to a particular boundary.	PPR-367
AllocMemMasked	Allocates a block of memory with specific bits set or cleared in its address.	PPR-369
AllocMemPages	Allocates whole pages of memory.	PPR-371
AllocMemTrack	Allocates a block of memory with size-tracking.	PPR-373
AllocMemTrackWithOptions	Allocates a block of memory with size-tracking and other options.	PPR-374
ControlMem	Controls memory permissions and ownership.	PPR-375
ControlMemDebug	Controls what MemDebug does and doesn't do.	PPR-377
CreateMemDebug	Initializes MemDebug, the Portfolio memory debugging package.	PPR-379
DeleteMemDebug	Releases memory debugging resources.	PPR-381
DumpMemDebug	Dumps memory allocation debugging information.	PPR-382
FreeMem	Frees memory that was allocated with <code>\f4AllocMem()</code> or <code>\f4AllocMemAligned()</code> or <code>\fP</code>	PPR-383
FreeMemPages	Frees memory that was allocated with <code>\f4AllocMemPages()</code> or <code>\fP</code>	PPR-384
FreeMemTrack	Frees memory that was allocated with <code>\f4AllocMemTrack()</code> or <code>\f4AllocMemTrackWithOptions()</code> or <code>\fP</code>	PPR-385
GetMemInfo	Gets information about available memory.	PPR-386
GetMemTrackSize	Gets the size of a block of memory allocated with <code>MEMTYPE_TRACKSIZE</code>	PPR-388
GetPageSize	Gets the size in bytes of a page of memory.	PPR-389
IsMemOwned	Determines whether a region of memory is owned by the current task. ...	PPR-390
IsMemReadable	Determines whether a region of memory is fully readable by the current task.	PPR-391
IsMemWritable	Determines whether a region of memory is fully writable by the current task.	PPR-392
RationMemDebug	Rations memory allocations to test failure paths.	PPR-393
ReallocMem	Reallocates a block of memory to a different size.	PPR-395
SanityCheckMemDebug	Checks all current memory allocations to make sure all the allocation cookies are intact.	PPR-397
ScavengeMem	Returns the current task's unused memory pages to the system page pool.	PPR-398

Messaging

CreateBufferedMsg	Creates a buffered message.	PPR-399
CreateMsg	Creates a standard message.	PPR-400
CreateMsgPort	Creates a message port.	PPR-401
CreateSmallMsg	Creates a small message.	PPR-402

CreateUniqueMsgPort.....	Creates a message port with a unique name.....	PPR-403
DeleteMsg.....	Deletes a message.....	PPR-404
DeleteMsgPort.....	Deletes a message port.....	PPR-405
FindMsgPort.....	Finds a message port by name.....	PPR-406
GetMsg.....	Gets a message from a message port.....	PPR-407
GetThisMsg.....	Gets a specific message.....	PPR-408
ReplyMsg.....	Sends a reply to a message.....	PPR-410
ReplySmallMsg.....	Sends a reply to a small message.....	PPR-412
SendMsg.....	Sends a message.....	PPR-413
SendSmallMsg.....	Sends a small message.....	PPR-415
WaitPort.....	Waits for a message to arrive at a message port.....	PPR-416

Miscellaneous

ReadHardwareRandomNumber.....	Gets a 32-bit random number.....	PPR-417
ReadUniqueID.....	Gets the system unique id.....	PPR-418

Semaphores

CreateSemaphore.....	Creates a semaphore.....	PPR-419
CreateUniqueSemaphore.....	Creates a semaphore with a unique name.....	PPR-420
DeleteSemaphore.....	Deletes a semaphore.....	PPR-421
FindSemaphore.....	Finds a semaphore by name.....	PPR-422
LockSemaphore.....	Locks a semaphore.....	PPR-423
UnlockSemaphore.....	Unlocks a semaphore.....	PPR-425

Signals

AllocSignal.....	Allocates signals.....	PPR-426
ClearCurrentSignals.....	Clears some received signal bits.....	PPR-428
FreeSignal.....	Frees signals.....	PPR-429
GetCurrentSignals.....	Gets the currently received signal bits.....	PPR-430
GetTaskSignals.....	Gets the currently received signal bits for a task.....	PPR-431
SendSignal.....	Sends signals to another task.....	PPR-432
WaitSignal.....	Waits until any of a set of signals are received.....	PPR-433

Tags

ConvertFP_TagData.....	Store a floating point value in a TagArg.....	PPR-434
ConvertTagData_FP.....	Extract a floating point value stored in a TagArg.....	PPR-435
DumpTagList.....	Prints the contents of a tag list.....	PPR-436
FindTagArg.....	Looks through a tag list for a specific tag.....	PPR-437
GetTagArg.....	Finds a TagArg in list and returns its ta_Arg field.....	PPR-438
NextTagArg.....	Finds the next TagArg in a tag list.....	PPR-439

Tasks

CreateModuleThread.....	Creates a thread from a loaded code module.....	PPR-441
CreateTask.....	Creates a task from a loaded code module.....	PPR-443
CreateThread.....	Creates a thread.....	PPR-445
DeleteModuleThread.....	Deletes a thread.....	PPR-448

DeleteTask	Deletes a task.....	PPR-449
DeleteThread	Deletes a thread.....	PPR-450
exit	Exits from a task or thread.....	PPR-451
FindTask	Finds a task by name.....	PPR-452
InvalidateFPState	Invalidates the current contents of the floating-point registers to prevent them from being saved during a context switch.....	PPR-453
RegisterUserException.....	Registers the exceptions to be handled by the current task.....	PPR-454
RegisterUserExcHandler	Registers an exception handler for the current task.....	PPR-455
Yield.....	Give up the CPU to a task of equal priority.....	PPR-457

Timer

AddTimerTicks	Adds two TimerTicks values together.....	PPR-459
AddTimes	Adds two time values together.....	PPR-460
CompareTimerTicks	Compares two timer ticks values.....	PPR-461
CompareTimes	Compares two time values.....	PPR-462
ConvertTimerTicksToTimeVal	Convert a hardware dependant timer tick value to a TimeVal.....	PPR-463
ConvertTimeValToTimerTicks	Convert a time value to a hardware dependant form.....	PPR-464
CreateTimerIOReq	Creates a timer device I/O request.....	PPR-465
DeleteTimerIOReq	Delete a timer device I/O request.....	PPR-466
SampleSystemTimeTT	Samples the system time with very low overhead.....	PPR-467
SampleSystemTimeTV	Samples the system time with very low overhead.....	PPR-468
SampleSystemTimeVBL	Samples the system VBL count with very low overhead.....	PPR-469
StartMetronome	Start a metronome counter.....	PPR-470
StartMetronomeVBL	Start a metronome counter.....	PPR-471
StopMetronome	Stop a metronome counter.....	PPR-472
StopMetronomeVBL	Stop a metronome counter.....	PPR-473
SubTimerTicks	Subtracts one TimerTicks value from another.....	PPR-474
SubTimes	Subtracts one time value from another.....	PPR-475
TimeLaterThan	Returns whether a time value comes before another.....	PPR-476
TimeLaterThanOrEqual	Returns whether a time value comes before or at the same time as another.....	PPR-477
TimerTicksLaterThan	Returns whether a time tick value comes before another.....	PPR-478
TimerTicksLaterThanOrEqual	Returns whether a time tick value comes before or at the same time as another.....	PPR-479
WaitTime	Waits for a given amount of time to pass.....	PPR-480
WaitTimeVBL	Waits for a given number of video fields to pass.....	PPR-481
WaitUntil	Waits for a given amount of time to arrive.....	PPR-482
WaitUntilVBL	Waits for a given vblank count to be reached.....	PPR-483

15

Portfolio Item Reference

Alias	A character string for referencing the pathname to a file or a directory of files.....	PPR-487
Attachment	The binding of a \f4Sample\fP or an \f4Envelope\fP to an \f4Instrument\fP or \f4Template\fP.....	PPR-488

AudioClock	Audio virtual timer item.	PPR-491
Bitmap	An Item describing an image buffer in RAM.	PPR-492
Cue	Audio asynchronous notification item.....	PPR-495
Envelope	Audio envelope.	PPR-496
ErrorText.....	A table of error messages.	PPR-501
File	A handle to a data file.	PPR-502
Instrument	DSP Instrument Item.	PPR-503
IOReq	An item used to communicate between a task and an I/O device.....	PPR-506
Knob	An item for adjusting an \f4Instrument\fP's parameters.	PPR-507
Locale	A database of international information.	PPR-509
Message	The means of sending data from one task to another.	PPR-510
MsgPort.....	An item through which a task receives messages.....	PPR-512
Probe.....	An item to permit the CPU to read the output of a DSP \f4Instrument\fP.....	PPR-513
Projector.....	An Item representing a particular physical display type.	PPR-514
Sample	A digital recording of a sound.....	PPR-519
Semaphore	An item used to arbitrate access to shared resources.	PPR-523
Task	An executable context.....	PPR-524
TEContext	An Item describing the context of the Triangle Engine.....	PPR-526
Template.....	A description of an audio instrument.	PPR-528
Tuning	An item that contains information for tuning an \f4Instrument\fP.....	PPR-529
View.....	An Item describing a displayed region of imagery.	PPR-531
ViewList.....	An anchor for a heirarchy of sub-Views.....	PPR-534

16

Requester Folio Calls

CreateStorageReq.....	Create a storage requester object.	PPR-537
DeleteStorageReq	Delete a storage requester object.....	PPR-539
DisplayStorageReq	Display a storage requester object to the user	PPR-540
ModifyStorageReq	Modify attributes of a storage requester object.	PPR-541
QueryStorageReq	Query the current state of certain attributes of a storage requester object.	PPR-543

17

SaveGame Folio Calls

LoadGameData	Loads a saved game file into memory.	PPR-547
SaveGameData	Stores a saved game file.....	PPR-549

18

Script Folio Calls

CreateScriptContext	Creates and initializes a ScriptContext structure.	PPR-553
DeleteScriptContext	Deletes a ScriptContext structure.	PPR-554
ExecuteCmdLine	Executes a shell command-line.....	PPR-555

19**Single Precision Math Library Calls****Conversion**

ceilf	round towards positive infinity.	PPR-559
fabsf	Floating Point Absolute Value.	PPR-560
floorf	round towards negative infinity.	PPR-561
fmodf	get the remainder of a division.	PPR-562
modff	get integer and fractional parts of a float.	PPR-563

Power

expf	Exponential Function.	PPR-564
frexpf	get fractional and exponent parts of a float.	PPR-565
ldexpf	multiply a float by a power of 2.	PPR-566
log10f	Base-10 Logarithm.	PPR-567
logf	Natural Logarithm.	PPR-568
powf	Power Function.	PPR-569
rsqrtf	Reciprocal Square Root Functions.	PPR-570
rsqrtff	Reciprocal Square Root Functions.	PPR-570
rsqrtfff	Reciprocal Square Root Functions.	PPR-570
sqrtf	Square Root Functions.	PPR-571
sqrtrf	Square Root Functions.	PPR-571
sqrtrff	Square Root Functions.	PPR-571

Trigonometric

acosf	Inverse Trigonometric Functions.	PPR-572
asinf	Inverse Trigonometric Functions.	PPR-572
atanf	Inverse Trigonometric Functions.	PPR-572
atan2f	arctangent of y/x.	PPR-573
cosf	Cosine and Sine Functions.	PPR-574
sinf	Cosine and Sine Functions.	PPR-574
coshf	Hyperbolic Functions.	PPR-575
sinhf	Hyperbolic Functions.	PPR-575
tanhf	Hyperbolic Functions.	PPR-575

3DO M2 Supplemental Portfolios Reference**1****Example Programs**

AutoMapper	Illustrates how to move and map the corners of a sprite.	MPR-3
autosizeview	Illustrates how to create a Bitmap and View automatically sized for the prevailing video mode.	MPR-4
basicview	Illustrates how to create a basic View.	MPR-5
compression	Demonstrates use of the compression folio.	MPR-6
DebugConsole	Demonstrates use of the debugging console.	MPR- 7

defaultport	Demonstrates using a task's default message port to communicate with a child thread.	MPR-8
fasttiming	Demonstrates how to use the high accuracy kernel timing services.	MPR-9
Ls	Displays the contents of a directory.	MPR-10
memdebug	Demonstrates the memory debugging subsystem.	MPR-11
metronome	Demonstrates how to use the metronome timer feature.	MPR-12
msgpassing	Demonstrates sending and receiving messages between two threads.	MPR-13
numinfo	Example program used to demonstrate 3DODebug functionality.	MPR-14
signals	Demonstrates how to use signals.	MPR-15
SpriteExample_1	Illustrates how to display a single sprite.	MPR-16
timerread	Demonstrates how to use the timer device to read the current system time.	MPR-17
timersleep	Demonstrates how to use the timer device to wait for an amount of time specified on the command-line.	MPR-18
Type	Types a file's content to the output terminal.	MPR-19
Walker	Recursively displays the contents of a directory, and all nested directories.	MPR-20

Audio

auto_bea	Automatic rhythm demo that uses lots of AIFF library samples.	MPR-21
capture_audio	Record the output from the DSP to a host file.	MPR-22
MarkovMusic	Constantly-varying environmentally-aware soundtrack.	MPR-23
minmax_audio	Measures the maximum and minimum output from the DSP.	MPR-25
playpimap	Listen to the instruments assigned in a pimap.	MPR-26
playsample	Plays an AIFF sample in memory using the control pad.	MPR-27
sfx_score	Uses libmusic.a score player as a sound effects manager.	MPR-28
simple_envelope	Simple audio envelope example.	MPR-30
ta_attach	Experiments with sample attachments.	MPR-31
ta_customdelay	Demonstrates a delay line attachment.	MPR-32
ta_envelope	Tests various envelope options by passing test index.	MPR-33
ta_pitchnotes	Plays a sample at different MIDI pitches.	MPR-34
ta_spool	Demonstrates the libmusic.a sound spooler.	MPR-35
ta_sweeps	Demonstrates adjusting knobs.	MPR-36
ta_timer	Demonstrates use of the audio timer.	MPR-37
ta_tuning	Demonstrates custom tuning a DSP instrument.	MPR-38
tj_canon	Uses the juggler to create and play a semi-random canon.	MPR-39
tj_multi	Uses the juggler to play a collection.	MPR-40
tj_simple	Uses the juggler to play two sequences.	MPR-41
tone	Simple audio demonstration.	MPR-42
tsp_algorithmic	Advanced sound player example showing algorithmic sequencing of sound playback.	MPR-43
tsp_rooms	Room-sensitive soundtrack example using advanced sound player.	MPR-45
tsp_spoolsoundfile	Plays an AIFF sound file from a thread using the advanced sound player.	MPR-47
tsp_switcher	Advanced sound player example that switches between sounds based on control pad input.	MPR-48
windpatch	Creates a "wind" sound effect using the audio folio's patch compiler.	MPR-49

Beep

tb_envelope	Trigger envelopes using the Beep folio.	MPR-50
tb_playsamp	Play a sample using the Beep Folio.	MPR-51
tb_spool	Spool audio from memory using the Beep folio.	MPR-52

EventBroker

cpdump	Queries the event broker and prints out a summary of what's connected to the control port.	MPR-53
focus	Talks to the event broker and switches the focus to a different listener.	MPR-54
lookie	Connects to the event broker and reports any events that occur.	MPR-55
luckie	Uses the event broker to read events from the first control pad.	MPR-56
maus	Uses the event broker to read events from the first mouse.	MPR-57

Frame2D

DrawLines	Illustrates three methods of drawing lines.	MPR-58
MoveSprite	Illustrates how to display, move, rotate and scale a single sprite.	MPR-59
Points	Illustrates how to draw points.	MPR-60
Rectangles	Illustrates how to move and map the corners of a sprite.	MPR-61
RenderOrder	Illustrates how to link sprites and control their rendering order.	MPR-62
SimpleSprite	Illustrates how to display a single sprite.	MPR-63
spin3d2d	Rotate 2d and 3d objects together.	MPR-64

Framework

sceneperf	Measure the performance of scenes.	MPR-65
-----------------	---	--------

graphics

m2perf	Measure the raw performance of the M2.	MPR-67
--------------	---	--------

Streaming

DataPlayer	A DataStream DATA subscriber example program.	MPR-73
EZFlxPlayer	A DataStream example program that plays synchronized video and audio.	MPR-74
PlaySA	A DataStream example program that plays streamed audio.	MPR-75
VideoPlayer	A DataStream example program that plays synchronized video and audio.	MPR-77

2

Shell Commands

AcroAdmin	Acrobat Filesystem Administration Utility.	MPR-81
AcroFormat	Format acrobat filesystems.	MPR-82
Clock	Gets/sets the date and time from the battery-backed clock.	MPR-83
Copy	Copies files or directories.	MPR-84
Delete	Deletes files and directories.	MPR-85
Dismount	Dismounts a file system.	MPR-86
dumplogs	Dump event logs to the debugging terminal.	MPR-87
expunge	Remove unused demand-loaded modules from memory.	MPR-88
FileAttrs	Gets or sets attributes of a file.	MPR-89
FindFile	Searches for a file, and prints its pathname.	MPR-90
FSInfo	Displays information on mounted file systems.	MPR-91

HW	Prints a list of all hardware resources currently in the system.....	MPR-92
Intl	Gets or sets the international settings of the machine.....	MPR-93
Items	Displays lists of active items.....	MPR-94
killtask	Remove an executing task or thread from the system.....	MPR-95
log	Control event logging.....	MPR-96
Ls	Displays the contents of a directory.....	MPR-97
MinimizeFS	Minimize filesystem/folio memory footprint.....	MPR-98
MkDir.....	Creates new directories.....	MPR-99
Mount.....	Mounts a file system.....	MPR-100
MountLevel.....	Displays or changes the current filesystem automounter level.....	MPR-101
Options	Controls various system run-time options.....	MPR-102
ProxyFile.....	Proxy a file so that it becomes a block-oriented device	MPR-103
RecheckFS.....	Recheck all filesystems to see if they are still online.....	MPR-105
Rename	Renames a file or directory.....	MPR-106
RmDir	Removes directories.....	MPR-107
setalias	Set a file path alias.....	MPR-108
setbg.....	Set the shell's default behavior to background execution mode.....	MPR-109
setcd.....	Set the shell's current directory.....	MPR-110
setfg	Set the shell's default behavior to foreground execution mode.....	MPR-111
setmaxmem.....	Set the amount of memory available in the system to the maximum amount possible.....	MPR-112
setminmem.....	Set the amount of memory available in the system to the minimum amount of memory guaranteed to be available in a production environment.....	MPR-113
setpri	Set the shell's priority.....	MPR-114
showavailmem.....	Display information about the amount of memory currently available in the system.....	MPR-115
showcd	Show the name of the current directory.....	MPR-116
showerror	Display an error string associated with a system error code.....	MPR-117
showfreeblocks.....	Show the contents of a memory list.....	MPR-118
showmemmap	Display a page map showing which pages of memory are used and free in the system, and which task owns which pages.....	MPR-119
showtask	Display information about tasks in the system.....	MPR-120
sleep	Cause the shell to pause for a number of seconds.....	MPR-121
SyncStress	Put some stress of task synchronization code to uncover title bugs.....	MPR-122
Type.....	Types a file's content to the output terminal.....	MPR-123
TypeIFF	Displays contents of an IFF file.....	MPR-124
Walker	Recursively displays the contents of a directory, and all nested directories.....	MPR-125
WriteMedia	Writes new raw data to media.....	MPR-126

Audio

audioavail	Show available audio resources in system.....	MPR-127
dspfadrs.....	Try out DSP instruments or patches.....	MPR-128
insinfo.....	Displays information about DSP Instruments.....	MPR-131
makepatch	Reads patch script language, writes binary patch file.....	MPR-132
playmf.....	Plays a standard MIDI file.....	MPR-137

3

MPEG Video Decompression Device

Using MPEG in a 3DO Title	MPR-140
Benefits of MPEG	MPR-140
Typical Data Flow	MPR-140
Video Synchronization	MPR-141
Using the Portfolio MPEG Device	MPR-141
Common Operations	MPR-142
Video Decoding Options	MPR-151
Output Modes	MPR-151
Other Commands	MPR-153
MPEG Still Image Decoding	MPR-155
MPEG Device Driver Memory Allocation	MPR-155

3DO M2 Lonely Chapters

dump3do	This command lets you examine the contents of 3DO executables and object modules.	LC-3
link3do	This command lets you build an M2 executable or DLL from object files, DLL modules, and libraries.....	LC-5

Assembly

__itof	Integer to float conversion.	LC-11
__utof	Integer to float conversion.	LC-11
DATAChunkify	A DataStream tool that prepares files for the DATA subscriber.....	LC-15

Volume 5:

D-C++ Language User's Manual

Introduction

Important features and extensions	1
High performance optimizations	1
Professional programming tools	2
Ease-of-use	3
Portability	3
This manual	3
Additional documentation	3
Document conventions	4

1 Installing the Compiler

Installation and compiler components	5
--	---

2 Accessing current and other versions of D-C++

Environment variables	8
Using D-C++ for different targets	9

3 Invoking the Compiler

The dplus command	11
Command line options	13
Compiler -X options	20
Examples of processing two source files	31
Compile and link	31
Separate compilation	32
Assembly output	33

4 Configuration File

How command lines, environment variables, and configuration files relate	35
Variables and precedence	35
D-C++ startup	35
Standard configuration files	37
UFLAGS1, UFLAGS2, DFLAGS, and command line options	39
The configuration language	39
Statements, options, and comments	39
Comments	40
String constants	40
Variables	40
Assignment statement	41
Error statement	42
Exit statement	42
If statement	42
Include statement	43

Print statement	43
Switch statement	43

5 Additions to ANSI C and C++

Predefined macros	45
Pragmas	45
inline	46
interrupt	46
no_alias	46
no_side_effects	47
pack	47
pure_function	48
no_return	49
section	49
use_section	49
asm and __asm	49
asm strings	49
asm macros	49
assert and unassert	52
Direct functions	52
Dynamic memory allocation with alloca	53
__ERROR__	53
extended	54
ident	54
#import	54
inline and __inline__	54
interrupt and __interrupt__	55
packed (max, min, byte-swapped) __packed__(max, min, byte-swapped)	55
pascal	56
sizeof	56

6 Optimization Hints

What to do from the command line	59
What to do with programs	61

7 The Lint Facility

8 Converting Existing Code to D-CC

Compilation problems	65
Execution problems	65
Functions with variable number of arguments	66

9 C++ Features and Compatibility

C++ features	69
Header files	69
Migration from C to C++	70
The D-CLASS C++ Library	71
Special C++ features	71

Construction / destruction of C++ static objects	71
Templates	71
Template instantiation	73
Exceptions	75
Array new and delete	77
Type identification	77
Dynamic casts in C++	77
C++ name mangling	77
Names used for operator encoding	80
Avoid setjmp and longjmp	81

A Compatibility modes: ANSI, PCC, and K&R C

B Compiler Limits

C Implementation-Defined Behavior

Translation	89
Environment	91
Library functions	92

D Error Messages

PowerPC Target User's Manual

1 Introduction

2 Target Dependent Command Line Options

Selecting a target	3
-X options	4

3 Compiler Components

The PPC directory	9
The target directories	9

4 Implementation Specific Behavior

Predefined macros and assertions	11
Pragmas	11
section use_section	11
Additions to ANSI C	17
long long	17

5 Internal Data Representation

Basic data types	19
Classes, structures, and unions	20
C++ classes	20
Pointers to members	22
Arrays	23
Bit fields	23

Byte ordering	23
Linkage and storage allocation	24

6 Calling Conventions

Stack layout	27
PowerPCEABI Stack Frame	27
PowerOpen Stack Frame	27
Argument passing	28
EABI Argument Passing	28
PowerOpen Argument Passing	30
C++ argument passing	30
Pointer to member as arguments and return types	31
Member function	31
Constructors and destructors	31
Result passing	31
Class return types	32
Register usage	32

7 Optimizations

Target independent optimizations	33
Tail recursion (0x2)	33
Inlining (0x4)	33
Argument address optimization (0x8)	34
Structure members to registers (0x10)	35
Local strength reduction (0x20)	35
Question - expression pop (0x40)	35
Assignment pop (0x80)	35
Simple branch optimization (0x100)	35
Space optimization (0x200)	35
Split optimization (0x400)	35
Constant and variable propagation (0x800)	36
Complex branch optimization (0x1000)	36
Loop strength reduction (0x2000)	36
Loop count-down optimization (0x4000)	37
Loop unrolling (0x8000)	37
Global common subexpression elimination (0x10000)	37
Undefined variable propagation (0x20000)	37
Unused assignment deletion (0x40000)	37
Minor transformations (0x80000)	37
Delayed register saving (0x100000)	37
Register coloring (0x200000)	37
Interprocedural optimizations (0x400000)	38
Remove entry and exit code (0x800000)	38
Use scratch registers for variables (0x1000000)	38
Extend optimization (0x2000000)	39
Loop statics optimization (0x4000000)	39
Loop invariant code motion (0x8000000)	40

Static function optimization (0x20000000)	40
PowerPCFeedback optimization (-Xfeedback)	40
Target dependent optimizations	40
PowerPCBasic reordering (0x1)	40
Branch to small code (0x4)	41
Delete no-ops (0x200)	41

8 Use in an Embedded Environment

Introduction	43
Compiler options for embedded development	43
User modifications	44
Start-up code	44
Global initialization	45
Hardware exception handling	46
Linker command file	46
Operating system calls	50
Character I/O	50
File I/O	50
Dynamic memory allocation	51
Miscellaneous functions	51
Stack checking	52
Position independent code (PIC)	52
Restrictions for position independent code	53
Communicating with the hardware	53
Mixing C and assembler functions	53
Using assembler code from within C programs	53
Accessing variables and functions at specific addresses	53
Re-entrant library functions	55
Command line arguments and environment variables	55
Profiling in an embedded environment	56
Support for multiple object formats	56

9 Example of Optimizations

PowerPC Target User's Manual

1 Introduction

2 Target Dependent Command Line Options

Selecting a target	3
-X options	4

3 Compiler Components

The PPC directory	9
The target directories	9

4 Implementation Specific Behavior

Predefined macros and assertions	11
--	----

Pragmas	11
section.....	11
use_section	11
Additions to ANSI C	17
long long	17

5 Internal Data Representation

Basic data types	19
Classes, structures, and unions	20
C++ classes	20
Pointers to members	22
Arrays	23
Bit fields	23
Byte ordering	23
Linkage and storage allocation	24

6 Calling Conventions

Stack layout	27
PowerPCEABI Stack Frame	27
PowerOpen Stack Frame	27
Argument passing	28
EABI Argument Passing	28
PowerOpen Argument Passing.....	30
C++ argument passing	30
Pointer to member as arguments and return types	31
Member function	31
Constructors and destructors	31
Result passing	31
Class return types	32
Register usage	32

7 Optimizations

Target independent optimizations	33
Tail recursion (0x2)	33
Inlining (0x4)	33
Argument address optimization (0x8)	34
Structure members to registers (0x10)	35
Local strength reduction (0x20)	35
Question - expression pop (0x40)	35
Assignment pop (0x80)	35
Simple branch optimization (0x100)	35
Space optimization (0x200)	35
Split optimization (0x400)	35
Constant and variable propagation (0x800)	36
Complex branch optimization (0x1000)	36
Loop strength reduction (0x2000)	36
Loop count-down optimization (0x4000)	37

Loop unrolling (0x8000)	37
Global common subexpression elimination (0x10000)	37
Undefined variable propagation (0x20000)	37
Unused assignment deletion (0x40000)	37
Minor transformations (0x80000)	37
Delayed register saving (0x100000)	37
Register coloring (0x200000)	37
Interprocedural optimizations (0x400000)	38
Remove entry and exit code (0x800000)	38
Use scratch registers for variables (0x1000000)	38
Extend optimization (0x2000000)	39
Loop statics optimization (0x4000000)	39
Loop invariant code motion (0x8000000)	40
Static function optimization (0x20000000)	40
PowerPCFeedback optimization (-Xfeedback)	40
Target dependent optimizations	40
PowerPCBasic reordering (0x1)	40
Branch to small code (0x4)	41
Delete no-ops (0x200)	41

8 Use in an Embedded Environment

Introduction	43
Compiler options for embedded development	43
User modifications	44
Start-up code	44
Global initialization	45
Hardware exception handling	46
Linker command file.....	46
Operating system calls	50
Character I/O	50
File I/O	50
Dynamic memory allocation	51
Miscellaneous functions.....	51
Stack checking	52
Position independent code (PIC)	52
Restrictions for position independent code	53
Communicating with the hardware	53
Mixing C and assembler functions	53
Using assembler code from within C programs	53
Accessing variables and functions at specific addresses	53
Re-entrant library functions	55
Command line arguments and environment variables	55
Profiling in an embedded environment	56
Support for multiple object formats	56

9 Example of Optimizations

Data Utilities User's Manual

1 Introduction

Document conventions	1
----------------------------	---

2 D-AR Archiver

Synopsis	3
Syntax	3
Description	3
Examples	5

3 D-BCNT Basic Block Counter

Synopsis	7
Syntax.....	7
Description	7
Files	7
Examples	8
Notes	8

4 D-DUMP File Dumper

Synopsis	9
Syntax	9
Description	9
Examples	11



3DO M2 2.0 Global Index

Version 2.0 – May 1996

Copyright © 1996 The 3DO Company and its licensors.

All rights reserved. This material constitutes confidential and proprietary information of The 3DO Company. This documentation is subject to a license agreement with The 3DO Company and may be used only by parties to such agreement. Use by any other persons, and/or for any purpose not expressly authorized by the agreement, is strictly prohibited.

3DO's LICENSOR(S) MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. 3DO'S LICENSOR(S) DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME JURISDICTIONS. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

Other brand or product names are the trademarks or registered trademarks of their respective holders.

3DO M2 Release 2.0 Global Index

Symbols

symbol DBG-5, DBG-19
. TSD-25
.spt file DBG-22
/remote> prompt DBG-8, DBG-9
-> symbol DBG-19, DBG-28
'A' path for the destination blender GPG-183

Numerics

2:1 3DO SQXD TSD-56
24-bit chunky RGB GSG-47
2D GPG-95
2D API, see also 2D Graphics Framework GPG-79
2D attributes
 and Triangle Engine GPG-95
2D bitmap rendering GPG-96
2D cel
 and sprite object GPG-86
2D cel, see also cel GPG-86
2D Framework object
 linking GPG-94
 rendering GPG-94, GPG-98
2D Framework objects
 names of GPG-81
2D Framework objects, see also 2D objects GPG-94
2D Framework, see also 2D Graphics Framework GPG-79
2D Graphics Folio
 responsibilities of GPG-1
2D Graphics Framework GPG-79
 and Triangle Engine GPG-79
2D Graphics Framework objects GPG-80

 linking GPG-82
 rendering GPG-81
2D Graphics Framework objects, see also 2D objects GPG-82
2D lines GPG-95
2D object
 drawing GPG-94
 rendering GPG-94
2D object primitives GPG-95
2D object, see also 2D framework object GPG-94
2D objects GPG-80, GPG-81
 linking GPG-82, GPG-94
 render list GPG-82
 rendering GPG-96
2D objects, see also 2D Framework objects GPG-94
2D point
 initializing GPG-121
2D points GPG-95
2D primitives GPG-95, GPG-3
2D rectangles GPG-95
2D rendering GPG-96
 and Triangle Engine GPG-96
2D sprite, see also sprite GPG-85
2D sprites GPG-80
2D triangle primitives GPG-96
2D triangles GPG-95
2-D, see also 2D GPG-1
3-2 PullDown VID-8

- 3-2 Pulldown DSG-10, VID-25
 - correcting incorrect field dominance VID-42
 - displaying reconstructed frames VID-42
 - identifying composite frames VID-40
 - overview VID-37
 - processing digitized materials VID-18
 - removing composite frames VID-40
 - saving processed movie VID-42
 - selecting field dominance VID-39
 - source requirements for using VID-39
 - synchronizing audio VID-42
- 3-2 Pulldown commands
 - Mark VID-40
 - Save as Pulldown VID-42
- 3-2 Pulldown tool VID-37
- 3D coordinates GPG-3
- 3D games
 - animating GPG-xvii
- 3D geometry GPG-1, GPG-2
- 3D heirarchical scenes and models GPG-2
- 3D model GPG-xviii
- 3D point
 - initializing GPG-121
- 3D points GPG-249
- 3D rendering GPG-2
- 3D rendering capabilities GPG-1
- 3D rendering library GPG-1
- 3D scene GPG-2
- 3D scenes
 - building and manipulating GPG-2
- 3D Sound MPG-117
- 3-D, see 3D GPG-1
- 3DO
 - 3dodebug folder DBG-8
- 3DO DataStream TSD-7
 - data flow overview DSG-21
 - debugging DSG-52
 - examples for use DSG-2
 - future development DSG-2
 - optimizing streaming DSG-52
 - overview DSG-1
 - threads DSG-20
- 3DO Debugger DBG-vii
 - examining structures DBG-32
 - Execution menu DBG-58, DBG-67
 - File menu DBG-52, DBG-62
 - preparing for launch DBG-2
 - Quick Start DBG-vii, DBG-1, DBG-25
 - Source window DBG-18
 - Special Mode DBG-54, DBG-63
 - starting DBG-2
 - Target menu DBG-59, DBG-68
 - Variables window DBG-33
 - View menu DBG-55, DBG-65
- 3DO debugger icon DBG-8
- 3DO Development Environment Installation Guide DBG-viii
- 3DO drive specifications CDM-40
- 3DO file format BNF GSG-50
- 3DO instrument TSD-18
- 3DO M2 Command List Toolkit GPG-82
- 3DO M2 debugger DBG-1
- 3DO M2 debugger, see also debugger DBG-1
- 3DO M2 debugger, see also M2 debugger DBG-1
- 3DO MPEGXpress VID-4
- 3DO Real-Time MPEG Encoding System VID-29
- 3DO Score File Format TSD-30
- 3DO Score File Format (3SF) TSD-30
- 3DO software GPG-3
- 3DO SoundHack DSG-12
- 3DO textures PRO-5, PRO-15, PRO-16
 - conversion PRO-10-PRO-12
- 3DODebug DBG-1
- 3DODebug, see also debugger DBG-2
- 3DODebug, see also M2 debugger DBG-2
- 3DODebug.Prefs file DBG-13
- 3DODebug.Prefs DBG-6
- 3DODebug.prefs DBG-55
- 3INS TSD-100
- 3SF TSD-32, TSD-40
 - sound design with TSD-32
- 3SF Architecture TSD-30, TSD-35
- 3SF architecture TSD-35
- 3SF file creation
 - from ARIA TSD-32
 - with MakeScore tool TSD-33
- 3SF file format TSD-89, TSD-90
 - all-in-one format TSD-32
- 3SF file playback
 - from ARIA TSD-33
 - from C or C++ program TSD-33
 - from PlayScore TSD-33
- 3SF format TSD-89, TSD-90, TSD-92
- 3SF tool set TSD-40
- 3SF tools TSD-42, TSD-43
 - DumpIFF TSD-42
 - MakeScore TSD-41
 - PlayScore TSD-43

4:1 ADPMC compression TSD-3, TSD-56

411 Help files

installing GSG-15

555 chunky RGB GSG-47

A

A MPG-94

AbortIO() PPG-113

About the 3DO M2 Development System GSG-1, GSG-8, GSG-1

absolute event times MPG-203

abstract View GPG-204

active window

closing DBG-52, DBG-62

AddHead() PPG-40

adding a node to a list PPG-40

AddTail() PPG-39, PPG-40, PPG-41

AddTimerTicks() PPG-123

AddTimes() PPG-118, PPG-122

AddTrace() DSG-57

AddViewToViewList function GPG-209, GPG-216, GPG-217

adjusting clipping planes GPG-271

Adobe After Effects VID-18, VID-19

Adobe AfterEffects VID-8

Adobe Photoshop, *see Photoshop*

Adobe Premier VID-8, VID-28

Adobe Premiere VID-24

ADPMC compression TSD-3

advanced sound player

adding sound files MPG-137

cleanup MPG-138

creating MPG-137

features MPG-136

looping and branching MPG-139

setting branches MPG-139

start playing sound MPG-137

start reading sound MPG-137

advanced sound player example MPG-136

aesthetic considerations TSD-7

AF MPG-80

AF_ENVF_LOCKTIMESCALE MPG-79, MPG-82

AF_TAG_ADDRESS MPG-49, MPG-75

AF_TAG_AMPLITUDE_FP MPG-36

AF_TAG_BASEFREQ MPG-49

AF_TAG_BASNOTE MPG-49

AF_TAG_CHANNEL MPG-49

AF_TAG_CLEAR_FLAGS MPG-52, MPG-82

AF_TAG_COMPRESSIONRATIO MPG-49

AF_TAG_COMPRESSIONTYPE MPG-49

AF_TAG_DELAY_LINE MPG-49

AF_TAG_DETUNE MPG-49

AF_TAG_FRAMES MPG-49, MPG-76

AF_TAG_HIGHNOTE MPG-49

AF_TAG_HIGHVELOCITY MPG-49

AF_TAG_HOOKNAME MPG-52

AF_TAG_INSTRUMENT MPG-52

AF_TAG_LOWNOTE MPG-49

AF_TAG_LOWVELOCITY MPG-49

AF_TAG_MAX MPG-70

AF_TAG_MIN MPG-70

AF_TAG_NAME MPG-70

AF_TAG_NUMBITS MPG-50

AF_TAG_NUMBYTES MPG-50

AF_TAG_PITCH MPG-36

AF_TAG_RELEASEBEGIN MPG-50, MPG-77

AF_TAG_RELEASEEND MPG-50, MPG-77

AF_TAG_RELEASEJUMP MPG-77

AF_TAG_RELEASETIME MPG-77

AF_TAG_SAMPLE MPG-52

AF_TAG_SAMPLE_RATE MPG-50

AF_TAG_SET_FLAGS MPG-52, MPG-82

AF_TAG_START_AT MPG-52

AF_TAG_SUSTAINBEGIN MPG-50, MPG-76

AF_TAG_SUSTAINEND MPG-50, MPG-76

AF_TAG_SUSTAINTIME MPG-76

AF_TAG_VELOCITY MPG-37

AF_TAG_WIDTH MPG-50

AfterEffects VID-25

AIFC MPG-46, TSD-100

AIFC file

converting to chunk file DSG-13

with SquashSnd TSD-25

AIFC FORM PPG-239

AIFC format MPG-45

AIFF VID-2, VID-7, VID-8, VID-11, GSG-58,

MPG-46, TSD-76, TSD-99

AIFF file

converting to chunk file DSG-13

editing TSD-4

from Red Book file TSD-4

generating with SoundHack TSD-56

preparing TSD-4

with SquashSnd TSD-25

AIFF samples TSD-18

AIFF-C GSG-58

algorithmic sound effects TSD-6

- aliasing GPG-158
 - avoiding (with mipmaps) CDM-21, GPG-154
- alignment
 - page GPG-201
 - specifying PPG-53
- AllocateContextInfo() PPG-264, PPG-265
- allocating a memory block PPG-53
- allocating buffers DSG-22
- allocating memory PPG-5
- AllocContextInfo() PPG-253
- AllocMem() PPG-5, PPG-52, PPG-53, PPG-59, PPG-61
- AllocMemAligned() PPG-54
- AllocMemBlocks() PPG-5, PPG-58
- AllocMemFromLists() PPG-5
- AllocMemMasked function GPG-201
- AllocMemPages() PPG-58
- AllocObject() MPG-198
- AllocSignal function GPG-81, GPG-82
- AllocSignal() CDM-14, PPG-13, PPG-84, PPG-26
- alpha
 - blended GPG-161
- alpha clamping GPG-190, GPG-193
- alpha clamping functions GPG-191
- alpha component GPG-259, GPG-124, GPG-149, GPG-186, GPG-191, GPG-192
- alpha component (of a texel) CDM-28, GPG-148
- alpha mask GPG-192
- alpha value GPG-186, GPG-191, GPG-192
 - dynamic range of CDM-34, GPG-153
- alpha value, see also alpha component CDM-28, GPG-148
- ambient GPG-277
- Ambient attribute GPG-112, GPG-114
- Ambient flag GPG-242
- ambient intensity GPG-124
 - setting GPG-124
- ambient light GPG-259, GPG-272, GPG-125
- ambient light color
 - obtaining GPG-124
- ambient lighting GPG-278
- Ambient material GPG-114
- Ambient property GPG-260
- amplitude value
 - setting up MPG-164
- analog joysticks PPG-199
- Analyze Only option TSD-65
- analyzing data DSG-7
- anchor PPG-38
- anchored list PPG-38
- and Graphics Pipeline GPG-106
- and overstrike prevention GPG-96
- angle
 - field of view GPG-273
 - field-of-view GPG-134
 - setting GPG-279
 - spotlight GPG-279
 - viewing GPG-272
- Angle attribute GPG-279
- angles
 - rotational GPG-241
- animating models GPG-280
- animating scenes GPG-xvii
- animation GPG-280, GPG-283
 - and characters GPG-280
 - and color GPG-281
 - and transforms GPG-280
 - converting DSG-12
 - creating GPG-xviii
 - frame by frame GPG-xviii
 - step by step GPG-xviii
- animation engine GPG-269, GPG-271, GPG-272
 - defined GPG-280
 - designating GPG-271
 - evaluting GPG-281
 - example of GPG-282
 - links GPG-282
 - links with characters GPG-272
 - obtaining array of GPG-271
- animation engine attributes GPG-281, GPG-283
- animation engine objects
 - evaluating GPG-282
- animation engines GPG-280, GPG-283
 - getting array of GPG-282
 - setting array of GPG-282
- animation functions GPG-281
- animation link GPG-269
- animation techniques GPG-xviii
- Animation type GPG-289
- anti-aliased font
 - creating FBR-1
- anti-aliasing resizing
 - edge crawl VID-15
- API
 - 2D, see also 2D Graphics Framework GPG-79
 - Graphics Framework GPG-227
- APIs
 - M2 GPG-2

-
- AppendPath() PPG-169
 - Apple HFS file CDM-26
 - Apple HFS format CDM-25
 - Apple MIDI Manager TSD-13, TSD-14
 - AppleScript PRO-dcccxiii, PRO-2
 - batch processing PRO-9, PRO-12
 - version supported PRO-3
 - application
 - quitting DBG-63
 - application blending
 - texture GPG-158
 - Application size limitations CDM-6
 - applications
 - multi-module PPG-289
 - application-specific geometry format GPG-293
 - applying tag argument values MPG-195
 - arbitrary quadrilateral GPG-86
 - ARIA
 - Comm3DO configuration with TSD-12
 - installing TSD-11
 - playing MIDI file TSD-18
 - required folders TSD-13
 - setup TSD-12
 - Array GPG-294
 - array
 - data area pointer for GPG-285
 - data areas of GPG-284
 - Graphics Framework GPG-283
 - in SDF GPG-247
 - link GPG-283
 - material GPG-247
 - material and texture GPG-256
 - object GPG-283
 - SDF GPG-294
 - texture GPG-151
 - array functions GPG-284
 - array of links GPG-272
 - Array type GPG-257
 - Array_Append function GPG-285
 - Array_Copy function GPG-285
 - Array_Create function GPG-284
 - Array_GetAt function GPG-285
 - Array_GetData function GPG-285
 - Array_GetFloat function GPG-286
 - Array_GetLong function GPG-285
 - Array_GetMaxSize function GPG-284
 - Array_GetObj function GPG-286
 - Array_GetShort function GPG-286
 - Array_GetSize function GPG-284
 - Array_Init function GPG-284
 - Array_SetAt function GPG-285
 - Array_SetData function GPG-285
 - Array_SetFloat function GPG-286
 - Array_SetLong function GPG-285
 - Array_SetMaxSize function GPG-285
 - Array_SetObj function GPG-286
 - Array_SetShort function GPG-286
 - Array_SetSize function GPG-284
 - arrays DBG-34
 - Graphics Framework GPG-283, GPG-286
 - in SDF files GPG-294
 - working with DBG-38
 - arrays of materials GPG-229
 - arrays of vertex data GPG-114
 - Ascent/Descent dialog FBR-5
 - aspect GPG-134
 - obtaining GPG-274
 - setting GPG-274
 - aspect adjustment
 - automatic GPG-271
 - Aspect attribute GPG-273
 - aspect ratio GPG-271, GPG-274
 - adjusting GPG-271
 - ASSERT() macros PPG-56
 - asterisk (*)
 - in Disassembly window DBG-20
 - AtomicClearBits() PPG-65
 - AtomicSetBits() PPG-65
 - AttachContextInfo() PPG-264, PPG-266
 - attachment
 - releasing independent MPG-57
 - stopping independent MPG-57
 - attachment item MPG-53
 - attachments MPG-20
 - attributes MPG-55
 - independent MPG-57
 - instrument-stopping MPG-56
 - linking MPG-58
 - modifying attributes MPG-55
 - setting attributes MPG-55
 - start independent MPG-56
 - starting dependent MPG-34
 - starting independent MPG-57
 - starting point for reverberations MPG-98
 - stopping MPG-35
 - tag args MPG-55
-

- attenuation GPG-279
- attribute
 - falloff GPG-277
 - omitting GPG-230
 - shininess GPG-259
- attribute functions
 - GP GPG-111
- attributes
 - audio MPG-88
 - character GPG-238
 - copying GPG-292
 - Graphics Pipeline GPG-111
 - item GPG-214
 - lighting GPG-278
 - Model GPG-257
 - of animation engine GPG-281, GPG-283
 - of GP objects GPG-109
 - of objects GPG-229
 - of objects in SDF files GPG-229
 - of objects in SDF files, see attributes GPG-230
 - pushing and popping GPG-111
 - reading MPG-88
 - rendering GPG-113
 - rendering-control GPG-117
 - scene GPG-269
 - setting MPG-88
- attributes of animation engine GPG-281
- attributes of characters GPG-238, GPG-239
- attributes of extended sprite objects GPG-88
- attributes of sprite objects GPG-86
- Audio MPG-75
- audio TSD-76
 - capture VID-17
 - compression DSG-9, DSG-12
 - converting to chunk file DSG-13
 - deciding on optimal compression DSG-10
 - generating DSG-10
 - Red Book MPG-217
- audio acquisition card VID-16
- audio attributes MPG-88
 - reading MPG-88
 - setting MPG-88
- audio clock MPG-40
 - checking current setting MPG-41
- audio clock rate MPG-168
 - setting MPG-168
- audio clock speed
 - setting MPG-157
- audio clock ticks DSG-7
- audio controller
 - subcodes PPG-213
- audio data TSD-7
- Audio folio MPG-10, MPG-19, MPG-127
 - closing MPG-39
 - opening MPG-21
- audio hardware MPG-9
- Audio Interchange File Format VID-2, GSG-58
- audio production TSD-4
- audio sample
 - attaching to instrument MPG-2
 - playing MPG-2
- audio samples
 - looping MPG-214
- audio *See alsound*
- audio signal MPG-61
- audio software MPG-10
 - writing MPG-1
- audio ticks MPG-40
- audio timer
 - for use with tick values MPG-189
- audio tools function summary TSD-75, TSD-76
- audio/video synchronization TSD-5
- AUDIO_TEMPLATE_NODE PPG-69
- AudioChunkifier VID-11
- audioclockchan DSG-15
- Audiomedia sound file TSD-54
- AUDIONODE PPG-69
- AudioSubscriber.c DSG-13
- AutoAdjust attribute GPG-268
- automatic aspect adjustment GPG-271
- avatars PPG-139
 - placement PPG-139
- avatars "could not assign" CDM-20
- axes GPG-241, GPG-242, TSD-97
 - global GPG-242, GPG-252
 - local GPG-242
 - world GPG-242
- axially aligned bounding box GPG-254
- axis
 - coordinate GPG-247, GPG-248
- axis attribute GPG-248
- AxisRotate GPG-247

B

- b PIMap flag TSD-19
- back clipping plane GPG-272
- BackColor attribute GPG-268, GPG-112, GPG-114
- background
 - of scene GPG-269, GPG-271
- background character GPG-270
- background color GPG-271
 - current GPG-271
 - getting GPG-271
 - of scene GPG-271
 - setting GPG-271
- Bands option TSD-64
- Bands pop-up TSD-68
- BannerScreen file CDM-13
 - requirements CDM-13
- base class GPG-289
- base class for viewable objects GPG-230
- base color GPG-259
- base-class functions GPG-290
- batch processing PRO-9, PRO-12
- Batt folio PPG-171
- battery-backed clock PPG-171
- BBL_MASelectConst GPG-188
- BDA GPG-256
- Beep Folio MPG-219
- bend value MPG-93
- bending a pitch MPG-176
- BendInstrumentPitch() MPG-177, MPG-93
- Best Quality settings VID-7
- Betacam SP VID-14
- bigfixed TSD-90
- Bi-linear filtering CDM-21
- bi-linear filtering CDM-26, GPG-147, GPG-155, GPG-157, GPG-192
- Binary MPG-103
- Binaural Filter TSD-57, TSD-58
- Binaural filter
 - creating stereo file TSD-57
 - using TSD-58
- Bio Bus GPG-256
- bitfields
 - SDF, see bitfields GPG-230
- Bitmap
 - creating (example program) GPG-203
 - rendering GPG-219, GPG-220
 - showing a subregion of GPG-208
- bitmap
 - GP GPG-111
- Bitmap buffer GPG-202
- Bitmap buffer memory GPG-202
- Bitmap buffers GPG-201
- Bitmap co-alignment GPG-201
- Bitmap dimensions GPG-201
- Bitmap item GPG-197, GPG-198, GPG-203
 - creating GPG-198
 - memory allocation for GPG-198
 - structure of GPG-198
- Bitmap Item fields GPG-199
- Bitmap Item properties GPG-200
- Bitmap page alignment GPG-201
- Bitmap parameters GPG-201
 - adjusting GPG-201
- Bitmap specification GPG-201
- Bitmap type GPG-204
- bitmaps GPG-111
 - allocating with GState object GPG-82
- blended alpha
 - obtaining GPG-161
- blender
 - destination, see also destination blender GPG-193
- blending
 - destination, see also destination blending GPG-173
 - texture application GPG-158
- blending channels GPG-189, GPG-190
- blending with a source frame buffer GPG-193, GPG-194
- block
 - drawing GPG-294
 - rectangular GPG-294
- Block Image file CDM-26
- Block Image icon CDM-27
- blockiness GPG-158
- block-oriented write PPG-127
- bm_Buffer field GPG-199
- bm_BufferSize GPG-202
- bm_BufferSize field GPG-199
- bm_BufMemCareBits GPG-201, GPG-202
- bm_BufMemCareBits field GPG-200
- bm_BufMemStateBits GPG-201, GPG-202
- bm_BufMemStateBits field GPG-200
- bm_BufMemType GPG-202
- bm_BufMemType field GPG-199
- bm_ClipHeight field GPG-199

- bm_ClipWidth field GPG-199
- bm_Height field GPG-199
- bm_Type field GPG-200
- bm_Width field GPG-199
- bm_XOrigin field GPG-199
- bm_YOrigin field GPG-199
- BNF GSG-50
- bool GPG-239
- bool type
 - SDF GPG-232
- border pixel GPG-96
- Bound attribute GPG-238, GPG-239, GPG-268
- boundary of scene GPG-272
- bounding GPG-238
- bounding box GPG-238, GPG-239, GPG-247, GPG-252, GPG-254, GPG-255, GPG-265, GPG-292
 - axially aligned GPG-254
 - bounding volume of GPG-239, GPG-271, GPG-274
 - computing GPG-293, GPG-139
 - coordinates GPG-270
 - getting bounding volume of GPG-265
 - of a scene GPG-270
 - rendering GPG-293
- bounding box operations GPG-253
- bounding box structure GPG-253
- bounding volume GPG-252, GPG-254, GPG-255, GPG-271
 - calculating GPG-292
 - computing GPG-144
 - of bounding box GPG-265
- bounds GPG-113
- box GPG-253
 - bounding, see also bounding box GPG-238, GPG-255, GPG-139
- box type
 - SDF GPG-232
- Box3 GPG-239, GPG-253
- Box3_Around function GPG-253
- Box3_Center function GPG-253
- Box3_Depth function GPG-253
- Box3_ExtendBox function GPG-253
- Box3_ExtendPt function GPG-253
- Box3_Height function GPG-253
- Box3_Set function GPG-253, GPG-109
- Box3_Transform function GPG-253
- Box3_Width function GPG-253
- Boyer-Moore algorithm PPG-65
- branching
 - and buffer size DSG-24
 - and key frames DSG-9
- break signals
 - sending PPG-135
- Breakpoint
 - removing DBG-29
 - setting DBG-29
- breakpoint
 - clearing DBG-19, DBG-20
 - initial DBG-16
 - setting DBG-28
- breakpoint symbol DBG-19
- breakpoints DBG-vii
 - in disassembly window DBG-20
 - using DBG-27
- Breakpoints command DBG-28, DBG-29
- BreakPoints menu item DBG-56, DBG-66
- BreakPoints window
 - removing a breakpoint DBG-29
 - setting a breakpoint DBG-29
 - using DBG-29
- Breakpoints window DBG-28, DBG-56, DBG-66
 - experimenting with DBG-29
- brightness
 - of lights GPG-229
- buffer
 - and Bitmap item GPG-198
 - Bitmap GPG-202
 - clearing GPG-112
 - flushing GPG-113
 - frame GPG-193
 - output GPG-192
- buffer list
 - creating DSG-22
- buffer memory
 - allocating (for Bitmaps) GPG-202
- buffering
 - double, see also double-buffering GPG-269
- buffers
 - allocating DSG-21
 - allocating DSG-22, DSG-9
 - deciding on number DSG-53
 - reuse DSG-26
 - starving buffers DSG-54
- built-in character types GPG-293
- built-in SDF types GPG-234

bumping
 dimension GPG-201
 bumping the Juggler MPG-204
 BumpJuggler() MPG-157, MPG-158, MPG-205
 buses
 M2 GPG-256
 busy-waiting PPG-4

C

CacheInfo structure PPG-62
 fields PPG-62
 Caches PPG-62
 calculations
 clipping GPG-293
 lighting GPG-125
 Call GPG-290
 Call_Class function GPG-290
 calling methods directly MPG-198
 Cam_Create function GPG-273
 Cam_GetAspect function GPG-274
 Cam_GetFOV function GPG-274
 Cam_GetHeight function GPG-274
 Cam_GetHither function GPG-274
 Cam_GetViewVol function GPG-273
 Cam_GetYon function GPG-275
 Cam_SetAspect function GPG-274
 Cam_SetFOV function GPG-273
 Cam_SetHeight function GPG-274
 Cam_SetHither function GPG-274
 Cam_SetViewVol function GPG-273
 Cam_SetYon function GPG-274
 Camera GPG-273
 camera GPG-252
 creating GPG-273
 defined GPG-272
 moving GPG-270
 orienting GPG-270
 perspective GPG-273
 SDF GPG-249
 searching for GPG-273
 setting the viewing volume GPG-273
 camera associated with scene
 obtaining GPG-270
 Camera attribute GPG-268
 camera attributes GPG-273
 camera class GPG-234
 camera position GPG-xviii
 cameras GPG-272, GPG-275, GPG-3, GPG-248
 capture masks PPG-176, PPG-184, PPG-188
 Catapult PPG-139
 dealing with large files CDM-19
 overview CDM-11
 catapultmegabytes variable CDM-18
 CBD2 compression VID-35
 CD Access Log CDM-19
 CD Access Log menu item DBG-60, DBG-69
 CD artwork CDM-32
 cd command DBG-14
 CD player MPG-8
 CDE GPG-256
 CD-ROM disc
 creating CDM-27
 CD-ROM image
 preparing CDM-13
 preparing optimized image CDM-16,
 CDM-17
 required files and folders CDM-12
 testing CDM-20
 CD-ROM mastering CDM-1
 application size limitations CDM-6
 error checking CDM-7
 mastering software CDM-4
 media CDM-4
 pathnames CDM-7
 reading integral number of disc blocks CDM-7
 recording speed CDM-24
 software requirements CDM-3
 tips CDM-39
 troubleshooting layout CDM-20
 CD-ROM mastering *See Also* Layout tool, Catapult
 CDM-10
 CD-ROM mechanism lifetime CDM-44
 CD-ROM recording station CDM-3
 CD-ROM space requirements CDM-6
 cdrom.tcl file CDM-13
 catapultmegabytes variable CDM-18
 editing for optimized image CDM-18
 editing for simple image CDM-14
 label variable CDM-18
 megabytes variable CDM-18
 cdrommaster folder CDM-12
 cel object
 and sprite objects GPG-86
 center
 geometric GPG-252
 ChangeDirectory() PPG-152
 ChangeItemOwner() PPG-11

ChangeScoreControl() MPG-167, MPG-175
 ChangeScorePitchBend() MPG-168, MPG-177
 ChangeScoreProgram() MPG-167, MPG-175
 changing TSD-50
 changing a program MPG-175
 changing current sound file TSD-48
 changing header information TSD-50
 changing pitch but not file length TSD-64
 channel header chunks DSG-42
 channel messages MPG-152
 channels
 blending GPG-189, GPG-190
 Char_Append function GPG-263, GPG-264
 Char_AxisRotate function GPG-243, GPG-247
 CHAR_BreadthFirst GPG-266
 Char_Clone function GPG-263, GPG-265
 Char_CloneDeep function GPG-263, GPG-265, GPG-292
 Char_Copy function GPG-292
 Char_Create function GPG-243, GPG-244, GPG-109
 Char_Delete function GPG-243, GPG-244
 CHAR_DepthFirst GPG-266
 Char_Display function GPG-243, GPG-245, GPG-292
 CHAR_DisplayAll GPG-292
 CHAR_DisplayNone GPG-292
 Char_First function GPG-263, GPG-264
 Char_ForAll function GPG-266
 Char_FuncCalcMatrix function GPG-292
 Char_FuncCopyDeep function GPG-292
 Char_FuncRender function GPG-290, GPG-292
 Char_FuncSelect function GPG-292
 Char_GetAt function GPG-263, GPG-264
 Char_GetBound function GPG-239, GPG-243, GPG-244, GPG-255, GPG-263, GPG-265, GPG-292
 Char_GetCenter function GPG-243, GPG-244, GPG-265, GPG-292
 Char_GetParent function GPG-263
 Char_GetSize function GPG-263, GPG-264
 Char_GetTransform function GPG-239, GPG-249
 Char_GetUserData function GPG-239
 Char_IsChild function GPG-263, GPG-265
 Char_IsCulling function GPG-239
 Char_IsParent function GPG-263, GPG-265
 Char_IsVisible function GPG-239
 Char_Last function GPG-264
 Char_LookAt function GPG-243, GPG-247

CHAR_Model GPG-289
 Char_Move function GPG-243, GPG-246, GPG-272
 Char_Next function GPG-266
 Char_Parent function GPG-265
 Char_Pitch function GPG-243, GPG-247
 Char_Print function GPG-243, GPG-245
 Char_PutAfter function GPG-263, GPG-264
 Char_PutBefore function GPG-263, GPG-264
 Char_Remove function GPG-263, GPG-264
 Char_Render function GPG-290
 Char_Reset function GPG-243, GPG-247
 Char_Roll function GPG-243, GPG-246
 Char_Rotate function GPG-243, GPG-247
 Char_Scale function GPG-243, GPG-247
 Char_SetTransform function GPG-249
 Char_SetUserData function GPG-240
 Char_SetVisible function GPG-239
 Char_Size function GPG-243, GPG-246
 CHAR_TopOnly GPG-266
 Char_TotalTransform function GPG-249
 Char_Translate function GPG-243, GPG-247
 Char_Turn function GPG-243, GPG-246
 Char_Yaw function GPG-243, GPG-247
 Character
 defining new subclass of GPG-289
 character
 appending GPG-264
 background GPG-270
 camera as GPG-272
 copying GPG-292
 displaying GPG-292
 dynamic
 character
 static GPG-269
 getting bounding box of GPG-265
 links with animation engines GPG-272
 manipulating GPG-241
 orienting GPG-240
 parent GPG-261
 removing from hierarchy GPG-264
 rendering GPG-292
 rotating GPG-240, GPG-246, GPG-247, GPG-248
 scaling GPG-249
 selecting for display GPG-292
 static GPG-269
 Superheroine GPG-262
 target GPG-272

- transformation of GPG-242
- character attributes GPG-238
 - listed GPG-239
- character axes
 - levels of GPG-242
- Character class GPG-295
- character class GPG-230, GPG-234
- character coordinates GPG-242
 - world coordinate system GPG-241
- character functions
 - global GPG-247
- character geometry GPG-262
- character hierarchy GPG-238, GPG-260, GPG-268
 - associating with a scene GPG-269
 - camera and GPG-272
 - child GPG-238, GPG-261
 - displaying GPG-292
 - iterator GPG-266
 - parent GPG-238, GPG-261
- character hierarchy functions GPG-264
- character hierarchy iterators GPG-266, GPG-267
- character iterator GPG-266
 - initializing GPG-266
- character moving GPG-240
- character operations
 - functions for GPG-243
 - geometric GPG-241, GPG-243
- character override GPG-292
- character reference GPG-261
- character rotation GPG-247
- character rotation functions GPG-247
- character scaling GPG-240
- character size
 - obtaining GPG-264
- character strings
 - working with PPG-224
- character transform GPG-292
- character transformation GPG-238, GPG-247
 - global GPG-242
- character transformation matrix GPG-248
- character transformation operations GPG-238, GPG-249
- character turn functions GPG-244
- Character type GPG-289
- character types GPG-293
 - built-in GPG-293
- Characters GPG-238
- characters GPG-230, GPG-232, GPG-238, GPG-254, GPG-3
 - and SDF files GPG-229
 - animation and GPG-280
 - attributes of GPG-238
 - collections of GPG-261
 - creating GPG-109
 - culling GPG-252
 - defined GPG-238
 - described geometrically GPG-256, GPG-260
 - geometric operations for GPG-240
 - in scenes GPG-248
- CheckIO() PPG-110
- CheckItem() PPG-74
- child tasks PPG-4, PPG-19
- children
 - displaying GPG-292
 - in character hierarchy GPG-261
- chunk DSG-42, TSD-92
 - and time information DSG-33
 - creating chunk file DSG-13
 - definition DSG-7
- Chunk header GSG-46
- Chunk order GSG-48
- Chunk size GSG-46
- Chunk structure definitions GSG-52
- chunk types DSG-21
- Chunk_ID GSG-46
- Chunks GSG-46
- Cinepak VID-21, VID-22, VID-23
- Cinepak compression
 - characteristics VID-46
 - setting data rate VID-25
- Cinepak conversion tool VID-24
- clamping
 - alpha GPG-190, GPG-191, GPG-193
- clamping color values GPG-125
- clamping TexBlend coordinates GPG-161
- class MPG-185
 - base GPG-289
 - camera, see also camera GPG-234
 - character GPG-230
 - character, see also character GPG-234
 - creating GPG-289
 - defining MPG-187
 - model, see also model GPG-234
 - scene, see also scene GPG-234
 - surface GPG-293
 - surface, see also surface GPG-234

- class definitions in SDF files GPG-294
- class descriptions
 - SDF GPG-263
- class ID GPG-287
- class inheritance GPG-289
- classes GPG-230
 - attributes of GPG-230
 - defining GPG-289, GPG-294
 - defining in SDF files GPG-294
 - Framework and SDF GPG-294
 - Graphics Framework GPG-233
 - in SDF files GPG-294
 - in SDF files, see classes GPG-230
 - registering GPG-233
 - SDF GPG-294
- classes of objects in SDF files GPG-229
- ClearBitRange() PPG-64
- ClearCurrentSignals() PPG-86
- clearing a scene GPG-271
- ClearRawFileError() PPG-149
- clip boundaries GPG-202
- clip region
 - resetting GPG-202
- clip window GPG-178
- Clipboard DBG-54, DBG-64
- clipping GPG-1, GPG-2, GPG-3, GPG-106, GPG-117
 - automatic GPG-272
 - window GPG-173, GPG-179
- clipping calculations GPG-293
- clipping plane GPG-135
 - back GPG-272
 - front GPG-272
 - hither GPG-272
 - hither, see also hither clipping plane GPG-272
 - yon GPG-272
 - yon, see also yon clipping plane GPG-272
- clipping planes GPG-135
 - adjusting GPG-271
- clipping window GPG-179, GPG-180
- Clockwise GPG-241
- Close command TSD-55
- Close menu item DBG-52, DBG-62
- CloseAudioFolio() MPG-180
- CloseCompressionFolio() PPG-229
- CloseDirectory() PPG-152
- CloseItem() PPG-75, PPG-11
- CloseMathFolio() MPG-180
- CloseModule() PPG-297
- CloseRawFile() PPG-151
- CloseSoundFile() MPG-135
- Cluster optimizer CDM-10
- CMD_BLOCKWRITE PPG-127
- CMD_STATUS PPG-126
- CMD_STREAMREAD PPG-132
- CMD_STREAMWRITE PPG-131
- co-alignment
 - Bitmap GPG-201
- collection
 - adding objects to MPG-196
 - creating and instance of MPG-195
 - messages for MPG-201
 - removing objects from MPG-197
 - setting up objects for MPG-195
 - setting variables with Tag Arg values MPG-196
- collection class MPG-188, MPG-190, MPG-195
- color
 - ambient light GPG-124
 - animating GPG-280
 - animation and GPG-281
 - background GPG-271, GPG-113
 - background, see also background color GPG-271
 - base GPG-259
 - computing GPG-124
 - diffuse GPG-259, GPG-124
 - emissive GPG-259
 - getting GPG-279
 - of 3D objects CDM-17, GPG-144
 - primitive CDM-34, GPG-158
 - RGB components of GPG-124
 - setting GPG-279
 - specular GPG-259
- color and light
 - combining GPG-124
- Color attribute GPG-279, GPG-280
- color component (of a texel) CDM-28, GPG-148
- color components
 - of a texel CDM-32, GPG-152
- color constants CDM-34, GPG-153
- Color lookup tables GSG-48, GSG-55
- color table CDM-33, GPG-152
- color type
 - SDF GPG-232
- color value
 - indexed GPG-149
 - literal GPG-149
 - of a texel GPG-151

- color value, see also color component CDM-28, GPG-148
- color values
 - clamping GPG-125
 - of texel GPG-149
- Color4 type GPG-260
- colors
 - blending GPG-158
 - computing GPG-125
 - equation for computing GPG-125
 - facet GPG-247
- Colors attribute GPG-136, GPG-140
- column vector GPG-120
- combining light and color GPG-124
- Comm3DO application
 - running with ARIA TSD-12
- Command + comma DBG-29
- Command + D DBG-36
- Command + G DBG-28
- Command + J DBG-19
- Command + M DBG-32
- Command + Return DBG-8
- Command + semicolon DBG-29
- Command + T DBG-9
- Command + U DBG-29
- command list buffers CDM-10, CDM-11, CDM-13, GPG-82
 - space CDM-13
- command list-buffers GPG-82
- command lists CDM-1
- commands
 - MPW DBG-9
- comment symbol (#) DBG-5, DBG-6
- communication among tasks PPG-13
- communications
 - intertask GPG-82
- compact disc player MPG-9
 - Red Book MPG-10
 - Yellow Book MPG-10
- CompareTimerTicks() PPG-123
- CompareTimes() PPG-122
- component
 - homogeneous GPG-121
- components
 - color (of a texel) CDM-32, GPG-152
 - of a texel CDM-28, GPG-148
 - operating-system GPG-1
- components of light GPG-258
- composite frames
 - aliasing VID-40
 - created from interlace fields VID-40
 - in film edited after Telecine processing VID-41
 - marking VID-40
 - storage considerations VID-42
 - Telecine process VID-38
- compress data PPG-228
- Compress() PPG-231
- compressed format TSD-48, TSD-49
- compressed sound TSD-3
- compressed texel GPG-148
- compressedVideo DSG-45
- compression
 - audio DSG-10
 - constant backgrounds VID-15
 - data CDM-34, GPG-153
 - edge crawl VID-15
 - eliminating noise VID-15
 - MovieCompress Quality slider VID-46
 - of texels CDM-28, GPG-148
 - pans and zooms VID-15
 - quality VID-46
 - solid colors VID-15
- compression engine
 - creating PPG-228
 - deleting PPG-229
 - feeding PPG-229
- Compression folio PPG-227
- compression folio
 - callback function PPG-230
 - closing PPG-229, PPG-230
 - convenience calls PPG-231
 - how it works PPG-228
 - how to use PPG-228
 - opening PPG-229
- compression options
 - MovieCompress VID-25
- computing colors GPG-125
- Concatenates GPG-135
- cone of illumination GPG-279, GPG-126
- configuration data block PPG-183
- configuration messages PPG-183, PPG-189
- ConfigurationRequest data structure PPG-188
- console
 - debugging PPG-311, PPG-312
- console output
 - performing PPG-312
- Constant linear velocity CDM-40

- constant registers CDM-28, GPG-148
- constants TSD-91
 - color CDM-34, GPG-153
- constructor
 - defining GPG-290
 - example GPG-290
 - of object GPG-290
- Content Library TSD-4, TSD-5
- Continue Process command TSD-70
- contribution
 - diffuse and specular GPG-125
- contribution equation GPG-126
- Control attribute GPG-282
- control chunk DSG-6
- Control menu TSD-70
 - Continue Process command TSD-70
 - Pause Process command TSD-70
 - Show Output command TSD-70
 - Show Spectrum command TSD-70
 - Stop Process command TSD-70
- Control Messages MPG-152
- control pad PPG-102
 - monitoring PPG-215
- control pad return data PPG-194
- control port PPG-102
- control port devices PPG-102
- control registers
 - and TEContext GPG-211
- control signal instruments MPG-23, MPG-24
- Control subscriber DSG-17
- controlling memory allocations PPG-230
- ControlLumberjack() PPG-320
- ControlMem() PPG-9, PPG-57, PPG-58
- ControlMemDebug() PPG-59
- ControlPadEventData data structure PPG-194
- convenience calls PPG-214
- conventions
 - typographical DBG-viii
- conversion tools GPG-256
- convert current sound file to TSD-49
- convert current sound file to compressed format TSD-49
- convert file PRO-9
 - 3DO texture PRO-10–PRO-12
 - batch processing PRO-12
 - formats PRO-10
 - Photoshop to Texture PRO-10, PRO-11, PRO-15, PRO-16
- convert sound file to TSD-48
- convert sound file to compressed format TSD-48
- convert to compressed format TSD-48
- Convert12TET_F16 MPG-94
- ConvertGregorianToTimeVal() PPG-173
- converting Apple sound resource to sound file TSD-56
- converting sound DSG-12
- converting sound file to Apple sound resource TSD-56
- converting sound to AIFF files TSD-56
- ConvertTimerTicksToTimeVal() PPG-123
- ConvertTimeValToGregorian() PPG-173
- ConvertTimeValToTimerTicks() PPG-123
- ConvertToMovie VID-24
 - where to find VID-24
- convolution
 - multiplying spectra TSD-58
- Convolution command TSD-58
- Convolution dialog
 - Filter Gain TSD-59
 - Impulse window TSD-59
 - Kilobytes Needed To Process File TSD-59
 - Length used TSD-59
 - Moving TSD-60
 - Ring Modulate TSD-59
- coordinate axis GPG-247, GPG-248
- coordinate system GPG-247, GPG-248
 - display GPG-204
 - for Views GPG-204
 - right-handed GPG-241
 - world GPG-133, GPG-134
 - world, see also world coordinate system GPG-241
- coordinates
 - 3D GPG-3
 - character GPG-242
 - display GPG-204
 - homogeneous GPG-249
 - pixel GPG-204
 - polar GPG-133
- copy function GPG-291
- copy object
 - creating PPG-167
 - deleting PPG-168
- copying attributes GPG-292
- copying characters GPG-292
- copying objects GPG-291

-
- count
 - reference, see reference-counting GPG-289
 - CounterClockwise GPG-241
 - counting
 - reference GPG-261, GPG-283, GPG-289
 - CPAK chunk DSG-17
 - CR register DBG-44
 - create TSD-32
 - create 3DO score file TSD-36
 - create instrument MPG-20
 - create score file TSD-32, TSD-33
 - create score file with TSD-33
 - CreateBufferedMsg(PPG-14
 - CreateBufferedMsg() PPG-183, PPG-90
 - CreateCompressor() PPG-228, PPG-230
 - CreateCopyObj() PPG-165, PPG-167
 - CreateCustomSignalIOReq() PPG-107
 - CreateDebugConsole() PPG-312
 - CreateDebugConsoleVA(PPG-312
 - CreateDebugConsoleVA() PPG-312
 - CreateDecompressor() PPG-228, PPG-229, PPG-230
 - CreateDeviceStackList() PPG-105
 - CreateDirectory() PPG-151
 - CreateEnvelope() MPG-83
 - CreateFileInDir() PPG-141
 - CreateIFFParser() PPG-251, PPG-254, PPG-256
 - CreateInstrument() MPG-28
 - CreateIOReq() PPG-116, PPG-104, PPG-107, PPG-110, PPG-73
 - CREATEIOREQ_TAG_SIGNAL PPG-106
 - CreateItem function GPG-197, GPG-198, GPG-199, GPG-205, GPG-207, GPG-210, GPG-215, GPG-223
 - CreateItem() PPG-69, PPG-70, PPG-73, PPG-10, PPG-11, PPG-34, PPG-20
 - CreateItemVA() PPG-72, PPG-34
 - CreateLumberjack() PPG-320
 - CreateMemDebug() PPG-59
 - CreateModuleThread() PPG-25
 - CreateMsg PPG-14
 - CreateMsg() PPG-73, PPG-14, PPG-89
 - CreateMsgPort() PPG-183, PPG-73, PPG-15, PPG-88
 - CreateObject() MPG-203
 - CreateScoreContext() MPG-159
 - CreateScriptContext() PPG-317
 - CreateSemaphore() PPG-73, PPG-78, PPG-79
 - CreateSmallMsg() PPG-14, PPG-90
 - CreateSoundFilePlayer() MPG-130
 - CreateStorageReq() PPG-283, PPG-285
 - CreateTask PPG-26
 - CreateTask() PPG-21, PPG-25
 - CREATETASK_TAG_STACKSIZE PPG-21
 - CreateThread(PPG-26
 - CreateThread() PPG-73, PPG-24, PPG-25, PPG-26
 - CreateTimerIOReq() PPG-117, PPG-118
 - CreateTuning() MPG-91
 - creating TSD-6
 - buffer lists DSG-22
 - control functions TSD-65
 - stereo file TSD-57
 - creating a collection MPG-195
 - creating a compression engine PPG-228
 - creating a juggler object MPG-165
 - creating a linked list PPG-39
 - creating a MIDI environment MPG-157, MPG-158
 - creating a score context MPG-159
 - creating a semaphore PPG-78
 - creating a virtual MIDI device MPG-153
 - creating algorithmic sound effects TSD-6
 - creating an object MPG-187
 - Creating CD-ROM disc for testing CDM-27
 - creating collections MPG-190
 - creating sequences MPG-190
 - cropping VID-18
 - cstring TSD-91
 - cstring31 TSD-91
 - CTRL chunk DSG-7
 - cube
 - drawing GPG-294
 - cue MPG-40
 - cull faces
 - getting GPG-113
 - setting GPG-113
 - CullFaces attribute GPG-112, GPG-114
 - culling GPG-239
 - Culling attribute GPG-238, GPG-239
 - culling characters GPG-252
 - culling functions GPG-113
 - culling triangles GPG-113
 - Current Data submenu DBG-53, DBG-63
 - current language
 - determining PPG-223
-

- current sound file
 - changing TSD-48
 - converting to AIFF format TSD-49
 - playing TSD-49
- custom devices PPG-102
- custom events PPG-185
- cylinder
 - drawing GPG-294
- Cylindrical type GPG-243

D

- d PIMap flag TSD-19
- data
 - examining DBG-40
 - obtaining GPG-129
 - vertex GPG-138
- data acquisition thread DSG-20, DSG-21
 - initializing DSG-27
- data analysis DSG-7
- data area (of a texel)
 - obtaining GPG-164
- data area pointer
 - for array GPG-285
- data areas of arrays GPG-284
- data array
 - allocating GPG-138
- data compression CDM-34, GPG-153
- data delay CDM-42
- data delivery rate CDM-42
- Data Directory path DBG-63
- data field GPG-229
- data fields
 - SDF GPG-295
- data flow overview DSG-21
- Data menu item DBG-56, DBG-66
- data organization TSD-91
- data rate
 - issues DSG-9
 - spikes DSG-53
- data rate issues CDM-41
- data rate spikes DSG-9
- data structure PPG-46
- data structures MPG-181
 - FileSystemStat PPG-125
 - IOReq PPG-112
 - Link PPG-49
 - ListAnchor PPG-49
 - ListenerList PPG-210
 - Locale PPG-218
 - NoteTracker PPG-39
 - NumericSpec PPG-221
 - PodData PPG-212
 - PodDescriptionList PPG-208
 - SetFocus PPG-211
 - TagArg PPG-70
 - VBlankTimeVal PPG-120
- data type TSD-90
- data types TSD-90, TSD-91
- Data window DBG-40, DBG-56, DBG-66
 - using DBG-40
- dataacqdelta pri DSG-15
- DATAChunkify DSG-13
- DataStreamer *See* 3DO DataStreamer
- DataStreamer tool VID-28
- date
 - Gregorian PPG-171
- date and time
 - Gregorian
 - converting to TimeVal PPG-173
 - reading PPG-172
 - setting PPG-172
 - validating PPG-174
 - TimeVal
 - converting to Gregorian PPG-173
- Date folio PPG-172
- DateSpec arrays PPG-223
- DBALUCNTL CDM-73
- DBAMULTCNTL CDM-69
- DBAMULTCONSTSSB0 CDM-70
- DBAMULTCONSTSSB1 CDM-71
- DBBMULTCNTL CDM-71
- DBBMULTCONSTSSB0 CDM-72
- DBBMULTCONSTSSB1 CDM-72
- DBCONSTIN CDM-69
- DBDESTALPHACNTL CDM-74
- DBDESTALPHACONST CDM-75
- DBDISCARDCONTROL CDM-64
- DBDITHERMATRIX CDM-75
- DBINTCNTL CDM-64
- DBL_AMultConstControl GPG-186, GPG-188
- DBL_ASelectConstComplement GPG-188
- DBLA_A constant GPG-189
- DBLA_A_AND_B constant GPG-189
- DBLA_A_OR_B constant GPG-189
- DBLA_AlphaClampForceTo0 constant GPG-191

-
- DBLA_AlphaClampForceTo1 constant GPG-191
 - DBLA_AlphaClampLeaveAlone constant GPG-191
 - DBLA_AlphaDestOut constant GPG-180
 - DBLA_ASelectTexAlpha constant GPG-183
 - DBLA_ASelectTexColor constant GPG-183
 - DBLA_B constant GPG-189
 - DBLA_BlendEnable constant GPG-179
 - DBLA_BSelectConstColor constant GPG-186
 - DBLA_BSelectSrcAlpha constant GPG-186
 - DBLA_BSelectSrcColor constant GPG-186
 - DBLA_BSelectTexColorComplement constant GPG-186
 - DBLA_DestAlphaSelectConst constant GPG-191
 - DBLA_DestAlphaSelectRBlend constant GPG-191
 - DBLA_DestAlphaSelectSrcAlpha constant GPG-191
 - DBLA_DestAlphaSelectTexAlpha constant GPG-191
 - DBLA_DiscardAlpha0 constant GPG-181
 - DBLA_DiscardRGB0 constant GPG-181
 - DBLA_DiscardSSB0 constant GPG-181
 - DBLA_DiscardZClipped constant GPG-181
 - DBLA_DitherEnable constant GPG-180
 - DBLA_DSBSelctConst constant GPG-182
 - DBLA_DSBSelctObjSSB constant GPG-182
 - DBLA_DSBSelctSrcSSB constant GPG-182
 - DBLA_MASelctBySrcSSB constant GPG-185
 - DBLA_MASelctByTexSSB constant GPG-185
 - DBLA_MASelctConst constant GPG-184
 - DBLA_MASelctConstComplement constant GPG-185
 - DBLA_MASelctSrcAlpha constant GPG-184
 - DBLA_MASelctSrcAlphaComplement constant GPG-184
 - DBLA_MASelctSrcColor constant GPG-184
 - DBLA_MASelctSrcColorComplement constant GPG-185
 - DBLA_MASelctTexAlpha constant GPG-184
 - DBLA_MASelctTexAlphaComplement constant GPG-184
 - DBLA_MBCstBySrcSSB constant GPG-188
 - DBLA_MBCstByTexSSB constant GPG-188
 - DBLA_MBSelectConst constant GPG-187
 - DBLA_MBSelectConstComplement constant GPG-187
 - DBLA_MBSelectSrcAlpha constant GPG-187
 - DBLA_MBSelectSrcAlphaComplement constant GPG-187
 - DBLA_MBSelectTexAlpha constant GPG-187
 - DBLA_MBSelectTexAlphaComplement constant GPG-187
 - DBLA_MBSelectTexColorComplement constant GPG-187
 - DBLA_NotA_AND_B constant GPG-189
 - DBLA_NotA constant GPG-189
 - DBLA_NotA_OR_B constant GPG-189
 - DBLA_NotB constant GPG-189
 - DBLA_NotB_AND_A constant GPG-189
 - DBLA_NotB_OR_A constant GPG-189
 - DBLA_OneOnEqual constant GPG-189
 - DBLA_OutputOne constant GPG-189
 - DBLA_PixOutOnEqualZ constant GPG-176
 - DBLA_PixOutOnGreaterZ constant GPG-176
 - DBLA_PixOutOnSmallerZ constant GPG-176
 - DBLA_RGBDestOut constant GPG-180
 - DBLA_SelectConstColor constant GPG-183
 - DBLA_SelectSrcColorComplement constant GPG-183
 - DBLA_SrcInputEnable constant GPG-180
 - DBLA_SrcSSBSelectsDestAlpha constant GPG-192
 - DBLA_TexSSBSelectsDestAlpha constant GPG-192
 - DBLA_WinClipInEnable constant GPG-179
 - DBLA_WinClipOutEnable constant GPG-179
 - DBLA_XOR constant GPG-189
 - DBLA_ZBuffEnable constant GPG-179
 - DBLA_ZBuffOutEnable constant GPG-179
 - DBLA_ZUpdateOnEqualZ constant GPG-176
 - DBLA_ZUpdateOnGreaterZ constant GPG-176
 - DBLA_ZUpdateOnSmallerZ constant GPG-176
 - DblAlpha0InputSelect GPG-183
 - DblAlpha0ClampControl attribute setting and getting GPG-191
 - DblAlpha1ClampControl GPG-191
 - DblAlpha1ClampControl attribute setting and getting GPG-191
 - DblAlphaFracClampControl GPG-192
 - DblAlphaFracClampControl attribute setting and getting GPG-192
 - DblALUOperation GPG-190
 - DblALUOperation attribute setting and getting GPG-190
 - DblAMultCoefSelect GPG-185
 - DblAMultCoefSelect attribute getting and setting GPG-184
 - DblAMultConstControl GPG-186
 - DblAMultConstControl attribute setting and getting GPG-185
 - DblAMultConstSSB0 GPG-185, GPG-186
-

- DblAMultConstSSB0 attribute
 - setting and getting GPG-185
- DblAMultConstSSB1 GPG-185, GPG-186
- DblAMultConstSSB1 attribute GPG-186
- DblAMultRtJustify attribute
 - setting and getting GPG-186
- DblARightJustify attribute CDM-61, GPG-175
- DblBInputSelect GPG-183
- DblBInputSelect attribute
 - setting and getting GPG-187
- DblBMultCoefSelect GPG-188
- DblBMultCoefSelect attribute
 - setting and getting GPG-188
- DblBMultConstControl GPG-188
- DblBMultConstControl attribute
 - setting and getting GPG-188
- DblBMultConstSSB0 GPG-188, GPG-189
- DblBMultConstSSB0 attribute
 - getting and setting GPG-188
- DblBMultConstSSB1 GPG-188
- DblBMultConstSSB1 attribute
 - setting and getting GPG-188
- DblBMultRtJustify attribute
 - setting and getting GPG-190
- DblBRightJustify attribute CDM-61, GPG-175
- DblDestAlphaConstSelect GPG-192, GPG-193
- DblDestAlphaConstSelect attribute
 - setting and getting GPG-192
- DblDestAlphaConstSSB0 GPG-192
- DblDestAlphaConstSSB0 attribute
 - setting and getting GPG-192
- DblDestAlphaConstSSB1 GPG-192
- DblDestAlphaConstSSB1 attribute
 - setting and getting GPG-193
- DblDestAlphaSelect GPG-192, GPG-193
- DblDestAlphaSelect attribute GPG-192
 - setting and getting GPG-192
- DblDestAlphaSelectConst GPG-192, GPG-193
- DblDiscard attribute GPG-182
 - setting and getting GPG-182
- DblDitherMatrixA attribute GPG-177
 - getting and setting GPG-178
- DblDitherMatrixB attribute GPG-177
 - getting and setting GPG-178
- DblDSBSelect
 - setting and getting GPG-182
- DblDSBSelect attribute
 - setting and getting GPG-182
- DblDSBSelect constant GPG-182
- DblDSBSelectConst GPG-182
- DblEnableAttrs GPG-178, GPG-179, GPG-180
- DblEnableAttrs attribute GPG-178, GPG-181
- DblEnableAttrs attributes CDM-63, GPG-178
- DblFinalDivide attribute CDM-61, GPG-175
 - setting and getting GPG-190
- DblIMASelectConst GPG-185
- DblIMASelectConstComplement GPG-185
- DblIMBSelectConst GPG-188
- DblIMBSelectConstComplement GPG-188
- DblPixOutOnEqualZ attribute GPG-175
- DblPixOutOnGreaterZ attribute GPG-175
- DblPixOutOnSmallerZ attribute GPG-175
- DblRGBConstIn attribute
 - setting and getting GPG-183
- DblSrcBaseAddr attribute
 - getting and setting GPG-193
- DblSrcPixels32Bit attribute
 - getting and setting GPG-193
- DblSrcXOffset attribute
 - setting and getting GPG-194
- DblSrcXStride attribute
 - getting and setting GPG-193
- DblSrcYOffset attribute
 - setting and getting GPG-194
- DblWinClipInEnable GPG-178, GPG-179, GPG-180
- DblWinClipOutEnable GPG-178, GPG-179, GPG-180
- DblXWinClipMax CDM-63, GPG-178, GPG-179
- DblXWinClipMax attribute GPG-179
- DblXWinClipMin CDM-63, GPG-178
- DblXWinClipMin attribute GPG-178
- DblYWinClipMax CDM-63, GPG-178, GPG-181
- DblYWinClipMax attribute GPG-180
- DblYWinClipMin CDM-63, GPG-178, GPG-180
- DblYWinClipMin attribute GPG-180
- DblZCompareControl attribute GPG-175, GPG-176
- DblZUpdateOnEqualZ attribute GPG-175
- DblZUpdateOnGreaterZ attribute GPG-175
- DblZUpdateOnSmallerZ attribute GPG-175
- DblZXOffset attribute GPG-175, GPG-176
- DblZYOffset attribute GPG-175, GPG-177
- DBSRCALPHACNTL CDM-74
- DBSRCBASEADDR CDM-66
- DBSRCNTL CDM-66
- DBSRCOFFSET CDM-67
- DBSRCXSTRIDE CDM-66

- DBSSBDSBCNTL CDM-68
- DBXWINCLIP CDM-65
- DBYWINCLIP CDM-65
- DBZCNTRL CDM-67
- DBZOFFSET CDM-68
- DCNTL CDM-76
- DCT VID-23
- DeBabelizer VID-25, VID-19
- DEBUG DSG-51
 - DDEBUG=1 DSG-51
 - DPRINT_LEVEL=1 DSG-51
 - DPRINT_LEVEL=2 DSG-51
- debug command DBG-14, DBG-15
 - arguments to DBG-15
- debug console link library PPG-311
- debug version of an M2 program DBG-5
- DEBUG_OPTIONS DBG-5
- DEBUGCONSOLE_TAG_HEIGHT tag PPG-312
- DEBUGCONSOLE_TAG_TOP tag PPG-312
- DebugConsoleClear() PPG-313
- DebugConsoleMove() PPG-313
- DebugConsolePrintf() PPG-313
- Debugger VID-55
- debugger DBG-vii, PRO-3
 - components of DBG-2
 - features of DBG-1
 - introducing DBG-1
 - languages DBG-vii
 - loading DBG-8
 - M2 DBG-vii
 - setting up DBG-2
 - starting DBG-8
 - using DBG-vii
- debugger icon DBG-8
- debugger preferences DBG-6
- debugger preferences file, see also preferences file
DBG-2
- debugger script DBG-6, DBG-7
- debugging console PPG-311, PPG-312
 - preparing for output PPG-312
- debugging session DBG-1
 - resuming DBG-26
 - setting up DBG-2
 - starting DBG-13
- DebugSample() MPG-54
- decision functions MPG-141
 - rules MPG-143
- decode timestamp DSG-43
- Decompress() PPG-231
- decompressing data PPG-228
- decompressing sound TSD-3
- decompression engine
 - creating PPG-229
 - deleting PPG-229
 - feeding PPG-229
- default context node PPG-241
- default light GPG-270
- deferred rendering GPG-143, GPG-228
- DefineClass() MPG-187
- defining a new class MPG-187
- definition file PPG-288, PPG-290
- deinterlace VID-25
- deinterlacing VID-18
- delay instrument MPG-94
- delay line MPG-94, MPG-95, MPG-96
 - calculating MPG-109, MPG-111
 - connecting MPG-96
 - creating MPG-95
 - deleting MPG-96
 - oscilloscope data MPG-99
- delay time
 - setting MPG-109
- delayed playback MPG-95
- DeleteCompressor() PPG-228, PPG-229
- DeleteCopyObj() PPG-165, PPG-168
- DeleteDebugConsole() PPG-313
- DeleteDecompressor() PPG-229
- DeleteDirectory() PPG-151
- DeleteEnvelope() MPG-85
- DeleteFile() PPG-141
- DeleteFileInDir() PPG-141
- DeleteIFFParser() PPG-254, PPG-255, PPG-257
- DeleteInstrument() MPG-38
- DeleteIOReq() PPG-114
- DeleteItem function GPG-215
- DeleteItem() PPG-75, PPG-12, PPG-21
- DeleteLumberjack() PPG-321
- DeleteMemDebug() PPG-59
- DeleteMemDebug() PPG-59
- DeleteScoreContext() MPG-179
- DeleteScriptContext() PPG-317
- DeleteSemaphore() PPG-79, PPG-81
- DeleteSoundFilePlayer() MPG-134
- DeleteStorageReq() PPG-283, PPG-286
- DeleteTree() PPG-168
- DeleteTuning() MPG-92
- deleting a semaphore PPG-80
- deleting the compression engine PPG-229

- dependency lists TSD-104
- destination blend attribute
 - and sprites GPG-87
- destination blend attribute settings
 - 2D GPG-95
- destination blender
 - 'A' path for GPG-183
 - attributes of GPG-193
 - final attributes of GPG-193
- destination blending GPG-106, GPG-173
 - described GPG-173
 - how it works GPG-173
- destination blending illustrated GPG-173
- destination frame buffer GPG-175, GPG-193
- destination select bit, see also DSB GPG-192
- destination-blend attributes
 - setting and getting GPG-181
- destroying an object MPG-187
- DestroyObject() MPG-187
- destructor
 - defining GPG-291
- DetachSample() MPG-54, MPG-85
- developer (dev) card DBG-2
- development (dev) card
 - booting DBG-6
- development card DBG-9
- device
 - opening PPG-105
- device drivers PPG-115, PPG-102, PPG-104
- device item PPG-137
- device item number PPG-106
- device stack PPG-105
- device stacks PPG-104
- devices PPG-101
 - control port PPG-102
 - daisy chain PPG-103
 - generic classes PPG-212
 - software PPG-103
- devices in 3DO system PPG-115
- dialog boxes
 - Directories | Setup DBG-11
 - target setup DBG-6
- Diaquest card VID-16
- dictionary
 - SDF GPG-232
- Diffuse color GPG-242
- diffuse color GPG-259, GPG-124
- Diffuse component GPG-242
- Diffuse flag GPG-242
- diffuse light GPG-259, GPG-125, GPG-126
- Diffuse material GPG-114
- Diffuse property GPG-260
- Digital Film card VID-16
- digital signal processor MPG-141, MPG-8, MPG-9
- digitized audio in DRAM MPG-9
- dimension bumping GPG-201
 - Bitmap GPG-201
- dimensions
 - of Bitmaps GPG-201
- direct digital input MPG-9
- direction
 - Z GPG-272
- direction of view GPG-133
- directional light GPG-276
- directional light source
 - creating GPG-279
- directional lighting GPG-276
- directories PPG-138
 - configuring DBG-4
 - renaming PPG-141
- Directories menu item DBG-53, DBG-63
- Directories submenu DBG-53, DBG-63
- Directories | Setup DBG-63
- Directories | Setup dialog box DBG-11
- directory
 - changing DBG-9, PPG-152
 - creating PPG-151
 - data DBG-63
 - deleting PPG-151
 - finding PPG-152
 - getting information PPG-130
 - opening and closing PPG-151
 - reading PPG-153
 - selecting DBG-63
 - source DBG-63
- directory entry PPG-153
- Directory functions PPG-138
- directory functions PPG-151
- Directory structure PPG-152
- directory tree
 - copying PPG-165
 - deleting PPG-168
- Disassembly menu item DBG-56, DBG-65

- Disassembly window DBG-19, DBG-56, DBG-65
 - clearing breakpoints DBG-21
 - features of DBG-20
 - opening DBG-21
 - popup menu in DBG-21
 - setting breakpoints DBG-21
 - setting breakpoints in DBG-20
 - setting PC DBG-21
- Disc seeks CDM-41
- Discrete Cosine Transform VID-23
- disjoint
 - triangles GPG-95
- DismountFileSystem() PPG-153
- DisownAudioClock() MPG-179
- display GPG-214, GPG-224
 - default GPG-224
 - optimizing compilation of GPG-224
 - types of GPG-224
- display architecture GPG-214
 - Graphics Folio GPG-214
- display compilation GPG-223
- display construction GPG-223
- display coordinate system GPG-204
- display coordinates GPG-204
- display locking GPG-225
- display logic GPG-256
- display signal GPG-217, GPG-219
- Display Signals GPG-217
- display signals
 - example code GPG-221
- display synchronization GPG-217
- Displayability property
 - Bitmap item GPG-200
- displaying a surface GPG-294
- displaying characters GPG-292
- DisplayStorageReq() PPG-283, PPG-285
- distortion
 - and textures CDM-19, GPG-145
 - linear GPG-86
- dither value GPG-177
- dithering GPG-173, GPG-177
- DLL symbols PPG-289
- DLLs PPG-287, PPG-309
- DLLs
 - advantages of using PPG-288
 - benefits of using PPG-288
 - creating and using PPG-288
 - linking PPG-295
- DLLs, see also dynamic-link library PPG-287
- DMA MPG-23
- Document menu
 - Convert to command PRO-15
- Document Proxy PRO-6
 - icon PRO-6
- Document windows PRO-6
- DolO() PPG-111
- doppler cue MPG-118
- double buffering GPG-237
- double-buffering CDM-14, GPG-223, GPG-269
 - and OrderViews function GPG-225
- Double-speed CD-ROM drive CDM-40
- downsampling VID-11
- drag and drop PRO-3
- DRAM PPG-52
- drawing attributes
 - GP GPG-113
 - primitive GPG-113
- drawing function
 - prototype of GPG-114
- drawing functions GPG-294
- Drift MPG-109
- Drive specifications CDM-40
- DSB GPG-193
- DSB (destination select bit) GPG-192
- DSConnect() DSG-27
- DSControl() DSG-62
- DSGoMarker() DSG-37
 - GOMARKER_ABSOLUTE DSG-37
 - GOMARKER_BACKWARD DSG-37
 - GOMARKER_FORWARD DSG-37
- DSP MPG-141
- DSP Audio VID-3
- DSP Audio Path VID-11
- DSP Designer sound file TSD-54
- DSP frame MPG-30
- DSP Resources MPG-117, MPG-118
- DSP ticks MPG-30
- DSP time units MPG-30
- DSP timing MPG-217
- DSStartStream() DSG-34
- DSStopStream() DSG-34
- DumpBitRange() PPG-66
- DumpIFF TSD-42
- DumpIFF tool TSD-42
- DumpLumberjackBuffer() PPG-322
- DumpMemDebug() PPG-59, PPG-60
- DumpNode() PPG-47
- DumpNode() PPG-48

dumpstream DSG-9
 dumpstream -stats DSG-52
 DumpTagList() PPG-36
 Duration attribute GPG-282
 Dynamic attribute GPG-268
 dynamic loading PPG-288, PPG-289
 dynamic voice allocation MPG-170
 dynamic-link libraries PPG-287, PPG-309

E

EA IFF 85 Standard PPG-237
 EB_IssuePodCmdReply PPG-214
 EB_MakeTableReply message PPG-200
 echo MPG-98
 edge crawl VID-15
 anti-aliasing resizing VID-15
 Edit Function dialog TSD-65, TSD-66
 creating control functions TSD-65
 default scaling functions TSD-66
 Output Waveform dialog TSD-66
 Phase Vocoder command TSD-65
 Pitch Scale function TSD-65
 Time Scale button TSD-65
 using TSD-65
 Edit menu TSD-56
 editing TSD-76
 AIFF files TSD-4
 Editing cdrom.tcl file for optimized image CDM-18
 Editing cdrom.tcl file for simple image CDM-14
 editing synthesized TSD-76
 editing synthesized audio TSD-76
 editing text DBG-54, DBG-64
 effects instruments MPG-23
 ElapsedTime attribute GPG-282
 electron beam GPG-258
 ellipsoid
 drawing GPG-294
 Emission material GPG-114
 Emission property GPG-260
 emissive color GPG-259
 emissive part of material GPG-125
 en GPG-229
 enableaudiomask DSG-29, DSG-15
 Enabled attribute GPG-279
 encoding
 run-length GPG-148
 Eng_Compute function GPG-282
 Eng_Eval function GPG-281
 Eng_FuncEval function GPG-293
 Eng_FuncStart function GPG-293
 Eng_FuncStop function GPG-293
 Eng_Reset function GPG-281
 Eng_Start function GPG-281
 Eng_Stop function GPG-281
 engine
 animation GPG-269, GPG-271, GPG-272,
 GPG-281, GPG-282, GPG-293
 evaluating GPG-293
 linking GPG-280
 starting
 animation engine GPG-293
 stopping GPG-293
 engine attributes GPG-281, GPG-283
 Engine class GPG-281, GPG-295
 engine evaluation function GPG-293
 engine object GPG-281
 engine override functions GPG-293
 engine overrides GPG-293
 engine state
 initializing GPG-293
 engine, see also animation engine GPG-282
 engines
 animation, see also animation engines
 GPG-280
 Engines attribute GPG-268
 EngLink object GPG-282
 EngLink structures
 computing GPG-283
 evaluating GPG-283
 Enter key DBG-9
 entry point
 DLL PPG-288
 enumerations
 in SDF files GPG-294
 SDF, see enumerations GPG-230
 envelopes MPG-14, MPG-20, MPG-74
 attaching to instrument MPG-83
 creating MPG-83
 deleting MPG-85
 detaching from instrument MPG-85
 EnvelopeSegment data structure MPG-75
 loops MPG-76
 points MPG-74
 properties MPG-74
 start point MPG-55
 starting dependent MPG-34

- Environment type GPG-243
- environment variables
 - setting DBG-12
- equation
 - contribution GPG-126
 - lighting and color GPG-125
- equations
 - transformation GPG-120
- error codes PPG-2
- ErrorProc TSD-44
- ErrorProce TSD-44
- ESCNTL CDM-77
- Evaluate button
 - in Variables window DBG-34
 - using DBG-34
- Evaluate text box
 - in Variables window DBG-39
 - using DBG-34
- event broker PPG-175, PPG-176
 - configuring PPG-178
 - connecting to PPG-178, PPG-183, PPG-214
 - convenience calls PPG-214
 - disconnecting PPG-206, PPG-207, PPG-216
 - features PPG-178
 - file system state data structure PPG-199
 - high-performance use PPG-199
 - reconfiguring PPG-206
 - reconfiguring connection PPG-206
 - reply to command PPG-214
 - trigger mechanism PPG-190
- event broker listeners
 - getting list of PPG-210
- event broker message types PPG-181
- event broker messages PPG-178
 - accompanying data structures PPG-182
 - configuration PPG-183
 - configuration reply PPG-189
 - flavors PPG-178
- event capture mask PPG-176
- event data
 - reading PPG-194
- event execution MPG-204, MPG-205
- event masks PPG-176
- event message data blocks PPG-190
- event messages
 - configuration PPG-189
 - reading PPG-191
 - retrieving and replying PPG-191
- event monitoring PPG-182, PPG-190
- event notifications PPG-190
- event queues PPG-184, PPG-191
- event trigger mask PPG-176
- event types PPG-185
- EventBrokerHeader data structure PPG-189, PPG-191
- EventFrame data structure PPG-191
- events PPG-176
 - custom PPG-185
 - monitoring PPG-176
 - non-UI PPG-185
 - specifying and monitoring PPG-176
 - system PPG-185
 - UI PPG-185
 - waiting for PPG-134
- examining data DBG-40
- examining structures DBG-32
- Example MPG-42
- example
 - advanced sound player MPG-136
 - Weaver script DSG-14
- example code
 - creating a view GPG-208
 - optimizing display signals GPG-221
 - reordering views in a ViewList GPG-216
- example function
 - reorder_array GPG-286
 - shuffling textures GPG-258
- example program
 - creating a Bitmap GPG-203
 - creating a sprite object GPG-100
- examples
 - setting up message port DSG-24
 - using SDF files GPG-249
- excludecatapult command CDM-18
- executable DBG-10
- Execute | Step In menu command DBG-29
- Execute | Step Over menu command DBG-29
- ExecuteCmdLine() PPG-316, PPG-317
- ExecuteCmdLineVA() PPG-316
- ExecuteModule() PPG-297
- executing a MIDI message MPG-172
- Execution menu DBG-58, DBG-67
 - menus
 - Execution DBG-16

Execution | Initial breakpoint in task menu item
menu items

Execution | Initial breakpoint in task
DBG-16

exit point

DLL PPG-288

expansion format of a texel
obtaining GPG-164

Export SND Resource command TSD-56

Export SND resource command TSD-56

exporting module

building PPG-293, PPG-294

ExpungeByAddress() PPG-299

ExpungeBySymbol() PPG-299

extended TSD-90

extended sprite

lighting effects on GPG-88

RGBA color values of GPG-88

shading effects on GPG-88

Z-values GPG-88

extended sprite object GPG-88

and 2D Graphics Framework GPG-80

attributes of GPG-88

defined GPG-80

extended sprite object, see also extended sprite
GPG-88

extended sprite object, see also ExtSpriteObj
GPG-88

extended sprite object, see also sprite GPG-88

extensibility of Graphics Framework GPG-287,
GPG-297

extension mechanism

Graphics Framework GPG-233, GPG-294

ExtSpriteObj GPG-81

ExtSpriteObj attributes GPG-88

ExtSpriteObject

defined GPG-80

EZFlix VID-21, VID-22, VID-24, VID-51, VID-3,
VID-10

EZFlixChunkifier VID-10, VID-48

F

-f PIMap flag TSD-19

f symbolic information DBG-5

F2_ColoredLines function GPG-95

F2_Draw function GPG-94

F2_DrawLine function GPG-95

F2_DrawList function GPG-94

F2_FillRect function GPG-95

F2_Point function GPG-82, GPG-95

F2_Points function GPG-95

F2_ShadedLines function GPG-95, GPG-96

F2_Triangles function GPG-95, GPG-96

F2_TriFan function GPG-95, GPG-96

F2_TriStrip function GPG-95, GPG-96

facet colors GPG-247

faceted primitive

defining GPG-247

faceted surface

defining GPG-246

FallOff attribute GPG-279

falloff attribute GPG-277

FeedCompressor() PPG-228, PPG-229

FeedDecompressor() PPG-229

FeedDecompressorEngine() PPG-228

feeding the compression engine PPG-229

feeding the decompression engine PPG-229

field dominance VID-39

field of view GPG-134

obtaining GPG-274

setting GPG-273

field of view angle GPG-273

field-of-view angle GPG-134

fields GPG-258

in Bitmap item GPG-199

file DSG-15

3DO texture PRO-2, PRO-5, PRO-15, PRO-16

allocating blocks for PPG-127

batch processing PRO-12

convert PRO-9, PRO-10

convert 3DO texture PRO-10-PRO-12

definition PPG-290

finding PPG-142

getting status PPG-126

marking end of PPG-128

preferences DBG-2

reading blocks PPG-128

reading data from PPG-128

setting attributes PPG-142

supported types PRO-2

writing blocks to PPG-127

file device PPG-115

file device driver PPG-123

file devices

communicating with PPG-123

File folio PPG-138

- file folio
 - examples PPG-154
- File formats GSG-45, GSG-46
- File functions PPG-138
- File menu DBG-10, DBG-52, DBG-62, TSD-54
 - Close command TSD-55
 - Export SND resource command TSD-56
 - Import SND resource command TSD-56
 - listen to Aiff file command TSD-56
 - Open Any command TSD-55
 - Open command PRO-14, TSD-54
 - Quit command PRO-14, TSD-56
 - Save A Copy command TSD-55, TSD-56
- file structure TSD-92, TSD-93, TSD-94, TSD-95, TSD-96, TSD-97, TSD-99, TSD-100, TSD-101, TSD-102, TSD-104, TSD-105
- file structure TSD-92
- file system PPG-137
 - finding PPG-154
 - getting status PPG-125
 - minimizing PPG-153
 - mounting and dismounting PPG-153
- file system interface PPG-137
- file type
 - setting PPG-129
- file types and extension TSD-89
- file types and extensions TSD-89
- file version
 - setting PPG-129
- file version chunk TSD-95
- file virtual block size
 - setting PPG-129
- File | Find utility DBG-15
- File | New menu item DBG-52, DBG-62
- FILECMD_ALLOCBLOCKS PPG-127
- FILECMD_READDIR PPG-130
- FILECMD_READENTRY PPG-130
- FileInfo structure PPG-150
- filemap.out file CDM-19
- Files
 - reading from CD-ROM CDM-7
- files PPG-138
 - locations of DBG-10
 - opening and closing PPG-141
 - parsing GPG-233
 - renaming PPG-141
 - storing in 3DODebug DBG-55
 - structure of PPG-138
- files in examples folder TSD-76
- filesystem
 - examples PPG-154
 - getting status PPG-125
- FileSystem Trace menu item DBG-60, DBG-69
- FileSystemStat structure PPG-125
- filler
 - minimizing DSG-52
- FillRect function
 - 2D GPG-96
- film
 - frame of GPG-xvii
- Filter window option TSD-64
- filtering
 - bi-linear, see also bi-linear filtering CDM-26, GPG-147, GPG-155
 - in texture-mapping CDM-19, GPG-145
 - linear, see also linear filtering CDM-24, GPG-147, GPG-155
 - mipmap CDM-21, GPG-154, GPG-156
 - nearest (point), see also nearest (point) filtering CDM-24, GPG-147, GPG-155
 - quasi tri-linear, see also quasi tri-linear filtering CDM-26, GPG-147, GPG-156
- filtering modes
 - mipmap CDM-21, GPG-147, GPG-155
- FindAndOpenItem() PPG-75
- FindAndOpenItemVA() PPG-75
- FindClearBitRange() PPG-65
- FindCollection() PPG-253, PPG-258
- FindContextInfo() PPG-254, PPG-266
- FindFileAndIdentify() PPG-142
- FindFileAndOpen() PPG-142
- FindFinalComponent() PPG-169
- finding a semaphore PPG-80
- FindItem() PPG-73, PPG-11
- FindMsgPort() PPG-189, PPG-96
- FindNamedItem() PPG-73, PPG-11, PPG-15
- FindNamedNode() PPG-45, PPG-47, PPG-48
- FindPropChunk() PPG-261, PPG-262
- FindSemaphore() PPG-81
- FindSemaphore() macro PPG-79
- FindSetBitRange() PPG-65
- FindTagArg() PPG-35
- FindTask() PPG-21
- FirstNode() PPG-43, PPG-44
- flags
 - MIDI PIMap TSD-19
- flanging MPG-109
- Flash ROM DBG-6

- float type
 - SDF GPG-232
- floating-point registers window DBG-57, DBG-66
- flow control instructions
 - INT CDM-9
 - JA CDM-9
 - JR CDM-9
 - PAUSE CDM-9
 - SYNC CDM-9
 - TXLD CDM-9
- FlushDCache() PPG-64
- FlushDCacheAll() PPG-64
- focus holder PPG-211
- folder
 - newview DBG-4
- folder setup TSD-13
- Folio
 - Graphics, see also Graphics Folio GPG-1
 - Graphics, see Graphics Folio GPG-195
- folio management PPG-2
- folios
 - Audio and Beep MPG-219
- Font Size pop-up menu FBR-2
- Font Style checkbox FBR-2
- for GPG-1
- ForAll GPG-266
- forking tasks PPG-5
- FORM chunk TSD-93
- form chunk TSD-93
- format
 - geometry GPG-293
- formatting currency PPG-226
- formatting dates PPG-224
- formatting numbers PPG-226
- FOV attribute GPG-273
- FOV, see also field of view GPG-134
- FP (floating point) Registers window DBG-57, DBG-66
- FP Registers window DBG-45
- FP Supervisor Registers window DBG-47
- frame GPG-281, GPG-xvii
 - rendering GPG-xviii
- frame buffer
 - clearing with GState object GPG-82
 - destination GPG-175, GPG-193
 - source GPG-193, GPG-194
- frame buffer bitmaps CDM-13
- frame rate
 - final VID-14
 - selecting VID-46
- frame rates GPG-258
- frame size
 - cropping VID-18
 - resizing VID-19
- frame-by-frame animation GPG-xviii
- frame-differencing VID-23
- FrameOptions attribute GPG-268
- FrameOptions setting
 - obtaining GPG-271
- frames GPG-258
- Framework
 - Graphics, see also Graphics Framework GPG-1
- framework animation engines GPG-280, GPG-283
- Framework array
 - elements of GPG-283
- Framework array, see also Graphics Framework array GPG-283
- Framework objects GPG-283
 - and SDF files GPG-229
- Framework, see also Graphics Framework GPG-227
- free memory pages PPG-52
- free() PPG-9
- FreeAmplitude() MPG-179
- FreeContextInfo() PPG-267
- freeing a memory block PPG-54
- FreeMem() PPG-9, PPG-53, PPG-54, PPG-59, PPG-61
- FreeObject() MPG-199
- free-signal mask PPG-14
- FreeSignal() PPG-14, PPG-86
- frequency MPG-72
- from ARIA TSD-32
- from C++ program TSD-33
- front clipping plane GPG-272
- frustum GPG-135
- FSUtils Folio PPG-165
- function
 - engine evaluation GPG-293
 - for copying objects GPG-291
 - rotating GPG-244
- function calls MPG-210
- functions
 - animation GPG-281
 - array GPG-284
 - camera GPG-273
 - drawing GPG-294
 - engine override GPG-293
 - Graphics Framework GPG-228

graphics utility GPG-294
 lighting GPG-279
 LightProp structure GPG-124
 overriding GPG-289, GPG-290
 functions for using rendering primitives GPG-115

G

-g build option DBG-5
 -g option DBG-5
 gain
 scaling MPG-119
 Gain Change command TSD-61
 Gain Change dialog TSD-61
 Gain/Reduction option TSD-68
 game data
 loading into memory PPG-278
 saving PPG-276
 Gate-Duck pop-up TSD-67
 general MIDI
 working with TSD-22
 Geo_Clear function GPG-139
 GEO_Colors GPG-139
 GEO_Colors style GPG-136
 Geo_CopyData function GPG-139
 Geo_Create function GPG-109, GPG-137
 Geo_CreateData function GPG-138
 Geo_Delete function GPG-137, GPG-138
 Geo_DeleteData function GPG-138
 GEO_Dynamic GPG-137
 GEO_Dynamic bit GPG-138
 GEO_Dynamic style GPG-137
 Geo_GetBound function GPG-139
 Geo_GetColor function GPG-138
 Geo_GetHeaderSize function GPG-138
 Geo_GetLocation function GPG-138
 Geo_GetNormal function GPG-138
 Geo_GetStyleName function GPG-138
 Geo_GetTexCoord function GPG-138
 Geo_Init function GPG-137
 GEO_Lines GPG-137
 GEO_Locations GPG-139
 Geo_MakeNormals function GPG-139
 GEO_Normals GPG-139
 GEO_Normals style GPG-136
 GEO_Points GPG-137
 Geo_Print function GPG-138
 GEO_QuadMesh GPG-137
 Geo_SetColor function GPG-138
 Geo_SetLocation function GPG-138
 Geo_SetNormal function GPG-138
 Geo_SetTexCoord function GPG-138
 GEO_TexCoords GPG-139
 GEO_TexCoords style GPG-136
 Geo_Transform function GPG-139
 GEO_TriFan GPG-137
 GEO_TriList GPG-137
 GEO_TriStrip GPG-137
 GEOF2_COLORS value GPG-96
 GEOF2_NULL value GPG-96
 GEOF2_TEXCOORDS value GPG-96
 geometric center GPG-252
 geometric character operations GPG-241, GPG-243
 geometric description
 of models GPG-263
 geometric models GPG-238
 geometric operations GPG-240
 listed GPG-240
 geometric primitive
 rendering GPG-117
 geometric shapes GPG-256, GPG-260
 geometric transformations GPG-129
 geometrical data GPG-109
 geometry GPG-232, GPG-256, GPG-3
 3D GPG-1, GPG-2
 and SDF files GPG-229
 character GPG-262
 matching textures and materials GPG-256
 of sprite objects GPG-93
 sharing GPG-256
 Surface GPG-257
 geometry format
 application-specific GPG-293
 Geometry functions GPG-137
 Geometry object
 copying GPG-139
 geometry of GP primitives GPG-118
 geometry of graphics primitives GPG-118
 geometry of primitives GPG-118
 geometry pipeline
 lighting and GPG-280
 Geometry Pipeline Transformations GPG-135
 geometry pipeline transformations GPG-135
 Geometry struct
 constructing GPG-137
 destroying GPG-138
 Geometry structure GPG-136
 initializing GPG-137

- Geometry structure header GPG-138
- geometry structures
 - creating GPG-109
- Get Info command FBR-5
- GetAudioDuration() MPG-41
- GetAudioItemInfo() MPG-55, MPG-88
- GetCacheInfo() PPG-62
- GetCompressorWorkBufferSize() PPG-230
- GetControlPad() PPG-215
- GetCurrentContext() PPG-268
- GetCurrentSignals() PPG-86
- GetDCacheFlushCount() PPG-63
- GetDecompressorWorkBufferSize() PPG-230
- GetDirectory() PPG-152
- GetIFFOffset() PPG-270
- GetInstrumentPortInfoByIndex() MPG-68
- GetInstrumentPortInfoByName() MPG-69
- GetMemInfo() PPG-55
- GetMemTrackSize() PPG-55
- GetMouse() PPG-215
- GetMsg() PPG-191, PPG-15, PPG-92
- GetNodeCount() PPG-46
- GetNodePosFromTail() PPG-46
- GetNthFromObject() MPG-197, MPG-202
- GetNumInstrumentPorts() MPG-68
- GetObjectInfo() MPG-199, MPG-200, MPG-201
- GetPageSize() PPG-5, PPG-56
- GetParentContext() PPG-268
- GetPath() PPG-169
- GetRawFileInfo() PPG-149
- GetScoreBendRange() MPG-177
- GetTag() PPG-35
- GetThisMsg() PPG-94
- getting filesystem status PPG-125
- Getting Started With 3DO M2 Release 2.0 DBG-3
- gfloat GPG-260
- GFX_BoundsEmpty GPG-144
- GFX_ClipIn GPG-112
- GFX_ClipOut GPG-112
- GFX_ClipPartial GPG-112
- Gfx_DrawBlock function GPG-294
- Gfx_DrawCylinder function GPG-294
- Gfx_DrawEllipsoid function GPG-294
- Gfx_DrawTorus function GPG-294
- GFX_Object GPG-289
- GFX_Surface GPG-289
- GfxObj class GPG-295
- GfxObj object GPG-287, GPG-289
- GfxObj override functions GPG-290
- GfxObj type GPG-289
- ghost
 - rendering CDM-33, GPG-153
- glasses controller
 - subcodes PPG-213
- glob patterns CDM-20
- global axes GPG-242, GPG-252
- global character functions GPG-247
- global numeric index TSD-100
- global operation GPG-241
- global origin
 - camera and GPG-272
- Go DBG-58, DBG-67
- Go menu item DBG-58, DBG-67
- GOTO options DSG-48
 - GOTO_OPTIONS_ABSOLUTE DSG-49
 - GOTO_OPTIONS_FLUSH DSG-49
 - GOTO_OPTIONS_MARKER DSG-49
- GP GPG-237
 - disabling GPG-117
 - enabling GPG-117
 - pushing and popping GPG-111
 - rendering GPG-111
 - switching between GPG-111
- GP attribute functions GPG-111
- GP capabilities
 - obtaining GPG-117
- GP drawing attributes GPG-113
- GP drawing primitives GPG-113
- GP functions GPG-106, GPG-109
 - general-purpose GPG-111
- GP lighting operations GPG-123, GPG-128
- GP object
 - deleting GPG-117
- GP object attributes GPG-109
- GP object functions GPG-109
- GP objects GPG-109
 - creating GPG-109
 - naming GPG-109
 - pushing and popping attributes of GPG-111
- GP primitives GPG-113
 - drawing GPG-113
 - geometry of GPG-118
- GP rendering attributes GPG-113
- GP rendering functions GPG-113, GPG-117
- GP rendering operations GPG-113
- GP rendering primitives GPG-114, GPG-115
- GP surface
 - deleting GPG-117

- GP, see also Graphics Pipeline GPG-227, GPG-105, GPG-111
- GP_ prefix GPG-109
- GP_Clear function GPG-94, GPG-112
- GP_ClearAll GPG-271
- GP_ClearAll function GPG-271
- GP_ClearHiddenSurf GPG-271
- GP_ClearHiddenSurf function GPG-271
- GP_ClearLights function GPG-124
- GP_ClearScreen GPG-271
- GP_ClearScreen function GPG-271
- GP_Clippping attribute GPG-117
- GP_Create function GPG-109, GPG-111
- GP_Delete function GPG-117
- GP_Disable function GPG-117
- GP_Draw function GPG-117
- GP_DrawLines function GPG-116
- GP_DrawPoints function GPG-117
- GP_DrawQuadMesh function GPG-116
- GP_DrawTriFan function GPG-115
- GP_DrawTriList function GPG-115
- GP_DrawTriStrip function GPG-116
- GP_Drawxxx functions GPG-143
- GP_Enable function GPG-117
- GP_Flush function GPG-94, GPG-113
- GP_GetAmbient function GPG-124
- GP_GetBackColor function GPG-113
- GP_GetCapabilities function GPG-117
- GP_GetCullFaces function GPG-113
- GP_GetDestBuffer function GPG-112
- GP_GetHiddenSurf function GPG-113
- GP_GetLight function GPG-124
- GP_GetMaterial attribute GPG-128
- GP_GetModelView function GPG-135
- GP_GetProjection function GPG-135
- GP_GetViewport GPG-114
- GP_GetZBuffer function GPG-112
- GP_IsInView function GPG-112
- GP_Lighting attribute GPG-117
- GP_Pop function GPG-112
- GP_Push function GPG-112
- GP_SetAmbient function GPG-124
- GP_SetBackColor function GPG-113
- GP_SetCullFaces function GPG-113
- GP_SetDestBuffer function GPG-111
- GP_SetHiddenSurf function GPG-113
- GP_SetLight function GPG-124
- GP_SetMaterial attribute GPG-128
- GP_SetTexBlend function GPG-166
- GP_SetViewport function GPG-113
- GP_SetZBuffer function GPG-112
- GP_Texturing attribute GPG-117
- grab MPG-69, MPG-89
- GrabKnob() MPG-89
- graphics
- 2D, see also 2D graphics GPG-1
 - sprite-based GPG-1
- graphics ASI GPG-256
- graphics capabilities
- M2 GPG-2
- Graphics Folio GPG-195, GPG-3
- defined GPG-3
 - described GPG-195, GPG-1
 - how applications communicate with GPG-195
 - opening GPG-81
 - responsibilities of GPG-1
- Graphics Folio API GPG-195
- Graphics Folio display architecture GPG-214
- Graphics Folio items GPG-197
- Graphics Folio tips and tricks GPG-224, GPG-225
- Graphics Folio View, see View GPG-204
- Graphics Framework GPG-227, GPG-298
- 2D, see also 2D Graphics Framework GPG-79
 - and Graphics Pipeline GPG-227
 - defined GPG-227, GPG-2
 - described GPG-2, GPG-106
 - extensibility of GPG-229, GPG-287, GPG-297
 - functions of GPG-228
 - job of GPG-1
 - roles of GPG-2
- Graphics Framework API GPG-227, GPG-106
- Graphics Framework array
- defined GPG-283
- Graphics Framework arrays GPG-283, GPG-286
- Graphics Framework classes GPG-233
- and SDF classes GPG-230
 - defining GPG-289, GPG-294
- Graphics Framework extension mechanism GPG-233, GPG-294
- Graphics Framework extensions in SDF files GPG-294
- Graphics Framework functions GPG-106
- Graphics Framework interface GPG-2
- Graphics Framework lighting operations GPG-275, GPG-280
- Graphics Framework objects GPG-106
- and SDF files GPG-229
 - listed GPG-231

- graphics hardware
 - accessing GPG-3
 - M2 GPG-255, GPG-2
- graphics item, see also item GPG-197
- graphics items GPG-197
- graphics library
 - M2 GPG-241
- Graphics Pipeline GPG-2, GPG-105, GPG-194
 - and Graphics Framework GPG-227
 - Graphics Framework GPG-106
 - and SDF files GPG-231
 - compared with Graphics Framework GPG-2
 - compared with Graphics Pipeline GPG-2
 - creating GPG-111
 - defined GPG-105
 - described GPG-2
 - how it works GPG-106
 - illustrated GPG-227
 - initializing GPG-111
 - job of GPG-3, GPG-105
 - operations of GPG-106
- Graphics Pipeline API GPG-105
- Graphics Pipeline attributes GPG-111
- Graphics Pipeline classes
 - and SDF classes GPG-230
- Graphics Pipeline functions
 - general-purpose GPG-111
- Graphics Pipeline objects
 - creating GPG-109
- Graphics Pipeline rendering operations GPG-109
- Graphics Pipeline, see also GP GPG-105
- graphics primitives
 - geometry of GPG-118
- graphics system
 - M2 GPG-195
- graphics utility functions GPG-294
- GregorianCalendar structure PPG-172
- grid object GPG-93
- GridObj GPG-93
- group associated with a scene
 - obtaining GPG-270
- Group of Pictures DSG-45
- GS CDM-15
- GS_BeginFrame() CDM-14
- GS_Create function GPG-82
- GS_Create() CDM-14
- GS_Delete() CDM-15
- GS_FreeBuffers() CDM-15
- GS_GetCurListStart() CDM-14

- gs_ListPtr field CDM-13
- GS_Reserve() CDM-13
- GS_Reserve() CDM-13
- gs_SendList CDM-13
- GS_SetList() CDM-14
- GS_SetVidSignal() CDM-14
- GS_WaitIO CDM-14
- GS_WaitIO() CDM-14
- GState functions GPG-82
- GState object GPG-81, GPG-82
 - and 2D programs GPG-82
 - and 3D programs GPG-82
 - creating GPG-82
 - initializing GPG-82
 - job and purpose of GPG-82

H

- h PIMap flag TSD-19
- Hack menu TSD-56
 - Gain Change command TSD-61
 - Header Change command TSD-50
 - Loops & Markers command TSD-51
 - Phase Vocoder command TSD-64
 - Phase Vocoder dialog TSD-65
 - Spectral Dynamics command TSD-66
 - Varispeed command TSD-68
- Handley, Maynard VID-28
- Hardware GSG-2
- hardware
 - for texture rendering GPG-147
 - graphics GPG-255, GPG-2
 - rendering GPG-3, GPG-152
- hardware devices PPG-102
- harmonic relations TSD-7
- hashmark (#) DBG-19
- Head seeks CDM-44
- Header Change command TSD-50
- header file
 - modifying markers TSD-51
- header files
 - audio/audio.h PPG-69
 - kernel/io.h PPG-107
 - kernel/kernelnodes.h PPG-70
 - kernel/types.h PPG-70
- header information TSD-50
- headerless file TSD-50, TSD-55
 - saving TSD-55

-
- height
 - obtaining GPG-274
 - of sprite object GPG-87
 - setting GPG-274
 - Height attribute GPG-273
 - heirarchical models
 - 3D GPG-2
 - heirarchical scenes
 - 3D GPG-2
 - heirarchy of views and viewLists GPG-214
 - help window DBG-9
 - hidden surfaces GPG-113
 - HiddenSurf attribute GPG-112, GPG-114
 - hierarchical file format (of SDF) GPG-3
 - hierarchies GPG-252
 - hierarchy GPG-252
 - character GPG-238, GPG-260, GPG-266, GPG-267, GPG-268, GPG-269, GPG-272, GPG-292
 - character, see also character hierarchy GPG-261
 - child characters and GPG-261
 - drawing GPG-269
 - multiple parents and GPG-261
 - object GPG-229
 - of ViewLists GPG-215
 - of Views GPG-215
 - parents and GPG-261
 - removing character from GPG-264
 - hierarchy functions
 - character GPG-264
 - Highest/Lowest Band option TSD-68
 - Hither attribute GPG-273
 - hither clipping plane GPG-272
 - getting GPG-274
 - setting GPG-274
 - homogeneous component GPG-121
 - homogeneous coordinates GPG-249
 - homogeneous point
 - initializing GPG-121
 - homogeneous vertices GPG-120
 - horizontal blank GPG-260
 - HW Reset menu item DBG-59, DBG-68
-
- I**
-
- I/O PPG-101, PPG-105, PPG-2
 - aborting PPG-113
 - finishing PPG-114
 - multiple operations PPG-113
 - preparing for PPG-105
 - I/O bandwidth limitation DSG-8
 - I/O operations
 - cleaning up after PPG-131
 - I/O process PPG-105
 - I/O request PPG-104
 - for rendering instructions GPG-211
 - I/O requests PPG-104
 - creating PPG-106
 - deleting PPG-114
 - reading PPG-112
 - return notification PPG-106
 - icon
 - 3DO debugger DBG-8
 - Icon folio PPG-271, PPG-272
 - icons
 - ICON FORM chunks PPG-272
 - loading into memory PPG-272
 - saving to a file PPG-274
 - unloading from memory PPG-274
 - uses PPG-271
 - ID TSD-91
 - class GPG-287
 - identifier
 - class GPG-287
 - ies GPG-242
 - IFF PPG-276
 - !(c) chunk PPG-249
 - !App chunk PPG-249
 - !Mod chunk PPG-249
 - !Rem chunk PPG-249
 - !URL chunk PPG-250
 - !Ver chunk PPG-249
 - CAT chunks PPG-247
 - collection chunks PPG-251, PPG-253, PPG-257
 - finding PPG-258
 - registering PPG-258
 - container chunk PPG-238
 - container chunks PPG-245, PPG-246
 - context info structures PPG-253
 - context stack PPG-240
-

- ContextInfo structures
 - allocating PPG-265
 - attaching PPG-266
 - deallocating PPG-267
 - finding PPG-266
 - inserting PPG-266
 - removing PPG-267
 - using PPG-264
- default context node PPG-264
- EAIFF 85 PPG-237
- entry and exit handlers PPG-263
- entry handlers PPG-252, PPG-263
- exit handlers PPG-264
- file format PPG-238
 - 3DO extension PPG-245
 - description PPG-243
- FORM chunks PPG-246
 - getting current context node PPG-268
 - getting parent context node PPG-268
 - getting seek position in stream PPG-270
 - global and local chunk ids PPG-247, PPG-248
 - goals PPG-239
 - LIST chunks PPG-247
 - local chunk PPG-238
 - parse operation
 - starting or continuing PPG-256
- parser structure
 - creating PPG-256
 - deleting PPG-256, PPG-257
- popping a context node PPG-269
- PROP chunks PPG-247, PPG-259
- property chunks
 - finding PPG-262
 - registering PPG-261
 - scoping rules PPG-259
 - vs. collection chunks PPG-261
 - vs. PROP chunks PPG-259
- pushing a context node PPG-268
- reading a chunk PPG-269
- reading and parsing a FORM PPG-250
- seeking in chunk PPG-270
- stop chunks PPG-252
 - registering PPG-262
- use of in 3DO M2 PPG-240, PPG-243
 - writing a chunk PPG-269
 - writing out a FORM PPG-254
- IFF folio PPG-237
 - capabilities PPG-240
 - description of functions PPG-255
 - purpose PPG-240
- IFF_PARSE_SCAN PPG-252
- IFF_SIZE_UNKNOWN_32 PPG-255, PPG-269, PPG-270
- IFF_SIZE_UNKNOWN_64 PPG-255, PPG-269, PPG-270
- IFFParser structure PPG-251
- IFFTypeID structure PPG-261
- Ignore Breakpoints menu item DBG-59, DBG-68
- Ignore DebugBreakpoint menu item DBG-59, DBG-68
- Ignore Printf menu item DBG-60, DBG-68
- illuminated objects GPG-124
- illumination
 - cone of GPG-279, GPG-126
- illumination, see also lights GPG-275
- image
 - rasterized GPG-257
- image buffer memory GPG-198
- image file *See* CD-ROM image CDM-12
- image measurement PRO-6, PRO-16
- image specification
 - source GPG-175
- Image Viewer PRO-5, PRO-6, PRO-11, PRO-16
- images
 - displaying GPG-195, GPG-3
- Import SND Resource command TSD-56
- IMPORT_FLAG flag PPG-292
- IMPORT_NOW flag PPG-291
- IMPORT_ON_DEMAND flag PPG-292
- ImportByAddress() PPG-298
- ImportByName() PPG-298
- importing a MIDI score MPG-155, MPG-165
- importing a MIDI score file MPG-158
- importing module
 - building and executing PPG-295
- Impulse Window option TSD-59
- incident diffuse light GPG-259
- incident light GPG-259
- indexed color value GPG-149
- infinity GPG-121

-
- inheritance
 - by SDF classes GPG-295
 - class GPG-289
 - multiple GPG-295
 - single GPG-295
 - InitEventUtility() PPG-214
 - InitGP function GPG-237
 - initial breakpoint DBG-16
 - Initial Breakpoint in task menu item DBG-59, DBG-68
 - initializing a linked list PPG-39
 - initializing engine state GPG-293
 - initializing the Juggler MPG-186
 - InitJuggler() MPG-158, MPG-186
 - InitScoreDynamics() MPG-159
 - InitScoreMixer() MPG-159
 - Input focus PPG-177
 - input focus PPG-177, PPG-210
 - InsertNodeAlpha() PPG-48
 - InsertNodeAlpha() PPG-41, PPG-47
 - InsertNodeFromHead() PPG-39, PPG-41, PPG-48
 - InsertNodeFromTail() PPG-39, PPG-48
 - InsertNodeFromTail() PPG-41
 - install TSD-48
 - InstallEntryHandler() PPG-252
 - InstallExitHandler() PPG-264
 - Installing GSG-8
 - installing ARIA TSD-11
 - instance MPG-185
 - instructions
 - rendering GPG-210, GPG-211
 - instrument MPG-2
 - loading MPG-2
 - start playing MPG-3
 - stopping MPG-6
 - instrument template MPG-21, MPG-39
 - instrument templates
 - assigning MPG-155
 - predefined MPG-22
 - InstrumentPortInfo MPG-68
 - Instruments MPG-22
 - instruments MPG-11, MPG-19
 - attachment MPG-53
 - connecting one to another MPG-32
 - creating MPG-28, MPG-30
 - deleting MPG-38, MPG-39
 - disconnecting MPG-33
 - elements of MPG-11
 - freeing MPG-38
 - knobs MPG-67
 - loading MPG-20, MPG-30
 - loading template MPG-26
 - playing MPG-20
 - preparing MPG-20
 - priority MPG-12
 - releasing MPG-37
 - resources MPG-30
 - starting MPG-34
 - stopping MPG-35, MPG-38
 - tag args MPG-35
 - tuning MPG-90
 - types MPG-22
 - use of by tasks MPG-12
 - using MPG-20
 - Instruments folder TSD-13
 - int16 TSD-90
 - int32 TSD-90
 - int8 TSD-90
 - integer type
 - SDF GPG-232
 - intensity
 - ambient GPG-124
 - intensity of spotlights GPG-277
 - interface
 - Graphics Framework GPG-2
 - interface ASIC GPG-256
 - interfilter CDM-25, CDM-26
 - intermediate region GPG-158
 - internal memory GPG-256
 - international folio
 - character strings PPG-224
 - determining current language PPG-223
 - formatting dates PPG-224
 - formatting numbers PPG-226
 - internationalization PPG-217
 - interpreter procedure MPG-190
 - writing MPG-192
 - interpreting a MIDI message MPG-172
 - InterpretMIDIEvent() MPG-157, MPG-167, MPG-172
 - InterpretMIDIMessage() MPG-157, MPG-167, MPG-170, MPG-172, MPG-175
 - intertask communication PPG-2, PPG-13
 - intertask communications GPG-82
 - Introduction CDM-2
 - inverse transform GPG-131
 - obtaining GPG-131
 - IO_QUICK PPG-111
-

ioi_Command PPG-119
 IOInfo PPG-105
 IOInfo data structure PPG-104, PPG-105, PPG-107
 setting values PPG-109
 IOInfo fields
 setting PPG-116
 IOInfo structure PPG-116
 fields PPG-108
 initializing PPG-107
 IOInfo.ioi_CmdOptions PPG-116
 IOInfo.ioi_Recv PPG-116
 IOInfo.ioi_Send PPG-116
 IOReq PPG-104, PPG-106
 creating PPG-73
 sending PPG-124
 IOReq data structure PPG-116, PPG-105, PPG-106, PPG-112
 IOReq structure PPG-105
 IOReqs PPG-104
 IRCAM/BICSF sound file TSD-54
 IsAutoAdjust attribute GPG-268
 IsBitClear() PPG-65
 IsBitRangeClear() PPG-65
 IsBitRangeSet() PPG-65
 IsBitSet() PPG-65
 IsEmptyList() PPG-43
 IsMemOwned() PPG-56
 IsMemReadable() PPG-56
 IsMemWritable() PPG-56, PPG-94
 IsNode() PPG-44
 IsNodeB() PPG-44
 IsVisible attribute GPG-268
 Item
 TEContext GPG-210
 item
 creating GPG-197
 modifying GPG-197, GPG-213
 TEContext GPG-210, GPG-211
 View GPG-204
 ViewList GPG-210
 item header TSD-99, TSD-100
 item headers TSD-99
 item ID number PPG-11
 item modification
 described GPG-213
 item names TSD-97
 item number PPG-72
 of a device PPG-106

item numbers TSD-97
 item numbers and item names TSD-97
 item types PPG-69, PPG-11
 ItemNode data structure PPG-68
 items
 absolute address of PPG-73
 changing ownership PPG-74
 changing priority PPG-74
 checking existence of PPG-74
 closing PPG-75
 convenience calls PPG-72
 creating PPG-67, PPG-69
 deleting PPG-75, PPG-12
 finding PPG-73
 graphics GPG-197
 Graphics Folio GPG-197
 handling PPG-10
 managing PPG-11
 opening PPG-69, PPG-75
 procedure for working with PPG-67
 shared resources PPG-10
 working with PPG-68
 items, managing PPG-67
 iterator
 character GPG-266
 iterator functions GPG-266
 iterators
 character hierarchy GPG-266, GPG-267

J

jack
 headphone MPG-9
 line-level output MPG-9
 JGLR_START_FUNCTION MPG-193, MPG-196
 JGLR_TAG_CONTEXT MPG-194, MPG-196
 JGLR_TAG_EVENT_SIZE MPG-193
 JGLR_TAG_EVENTS MPG-193
 JGLR_TAG_INTERPRETOR_FUNCTION
 MPG-193
 JGLR_TAG_MANY MPG-193
 JGLR_TAG_MAX MPG-193
 JGLR_TAG_MUTE MPG-194
 JGLR_TAG_REPEAT_DELAY MPG-194,
 MPG-196
 JGLR_TAG_REPEAT_FUNCTION MPG-193,
 MPG-196
 JGLR_TAG_SELECTOR_FUNCTION MPG-194
 JGLR_TAG_START_DELAY MPG-194, MPG-196

JGLR_TAG_STOP_DELAY MPG-194, MPG-196
 JGLR_TAG_STOP_FUNCTION MPG-194,
 MPG-196

joystick data PPG-198

JString PPG-218

juggle class MPG-188

Juggler MPG-183

control calls MPG-210

event execution process MPG-205

example of using events MPG-206

initializing MPG-186

overview of MPG-203

terminating MPG-206

using to play collections MPG-203

using to play sequences MPG-203

Juggler object

collection MPG-4

sequence MPG-4

K

kernel

high level functions PPG-1

page allocation PPG-6

understanding PPG-1

keyboard PPG-102

Kill Stopped Task menu item DBG-58, DBG-67

KillEventUtility() PPG-216

killtask command DBG-17

Kilobytes Needed To Process file option TSD-59

Kind attribute GPG-279

knobs MPG-11, MPG-12, MPG-20, MPG-32,
 MPG-33, MPG-67

changing parameters MPG-12

checking parameters MPG-13

definition of MPG-12

grab MPG-67, MPG-69, MPG-89

parameters MPG-70

releasing MPG-73

tweak MPG-67

L

-l PIMap flag TSD-19

label variable CDM-18

languages DBG-vii

LastNode() PPG-44

launchme CDM-12

LAUNCHME.M2 PPG-20

layout script CDM-13

Layout tool CDM-10

cluster optimizer CDM-10

nonoptimized CDM-10

troubleshooting CDM-20

LeftEdge GPG-210

Length Used option TSD-59

LERP (linear interpolation) GPG-158

LERP values

getting and setting GPG-171

level CDM-20, GPG-147

level of detail (LOD) CDM-20, GPG-147

level of detail, see also LOD CDM-20, GPG-147

library functions

importing a MIDI score file MPG-149

library modules

DLL modules PPG-288

licks TSD-97

light GPG-252

ambient GPG-259, GPG-277, GPG-125

ambient, see also ambient light GPG-125

ambient, see also ambient light GPG-272

approximating GPG-258

blocking GPG-124

color of GPG-279

components of GPG-258

creating GPG-279

defaultl GPG-270

diffuse GPG-259, GPG-126

direction of GPG-259, GPG-123

directional, see also directional light GPG-276

disabling GPG-280

enabling GPG-280

incident GPG-259

point GPG-277

point, see also point light GPG-277

position of GPG-123

properties of GPG-229

radiating GPG-124

SDF GPG-249

see also spotlight GPG-279

spot GPG-277

spotlight GPG-277

light and color

combining GPG-124

Light attribute GPG-268

light color

ambient GPG-124

- light gun
 - subcode PPG-214
- Light object GPG-280
- light rays
 - parallel GPG-276
- light source
 - adding GPG-124
- light sources GPG-259, GPG-123
- Light_Create function GPG-279
- LIGHT_Directional GPG-279
- LIGHT_Directional type GPG-276
- Light_GetAngle function GPG-279
- Light_GetColor function GPG-279
- Light_GetFallOff function GPG-279
- Light_GetKind function GPG-279
- Light_GetProp function GPG-280
 - properties
 - of light GPG-280
- Light_IsEnabled function GPG-280
- LIGHT_Point GPG-279
- LIGHT_Point type GPG-277
- Light_SetAngle function GPG-279
- Light_SetColor function GPG-279
- Light_SetEnabled function GPG-280
- Light_SetFallOff function GPG-279
- Light_SetKind function GPG-279
- LIGHT_SoftSpot GPG-279
- LIGHT_SoftSpot type GPG-277
- LIGHT_Spot GPG-279
- LIGHT_Spot type GPG-277
- LightGunData PPG-197
- lighting GPG-1, GPG-2, GPG-3, GPG-106, GPG-117, GPG-123, GPG-128
 - ambient GPG-278
 - approximating GPG-258
 - enabling GPG-124
 - managing GPG-123
 - two-sided GPG-128
- lighting a scene GPG-270
- lighting and color equation GPG-125
- lighting attributes
 - listed GPG-278
- lighting calculations GPG-124, GPG-125
- lighting conditions
 - and computing colors GPG-125
- lighting effects
 - on extended sprite objects GPG-88
- lighting functions GPG-279
- lighting operations GPG-123, GPG-128
 - Graphics Framework GPG-275, GPG-280
- lighting position GPG-xviii
- LightProp
 - obtaining GPG-124
- LightProp struct
 - architecture of GPG-123
- LightProp structure GPG-280
- LightProp structure functions GPG-124
- lights GPG-275, GPG-280, GPG-3, GPG-248
 - described GPG-275
 - removing GPG-124
 - sources of GPG-275
- line of sight GPG-133, GPG-134
- line primitives
 - 2D GPG-95
- line set
 - drawing GPG-116
- linear distortion
 - in sprite object GPG-86
- Linear filtering CDM-21
- linear filtering CDM-24, GPG-147, GPG-155, GPG-157
 - equation for CDM-25, GPG-155
- linear interpolation, see also LERP GPG-158
- lines GPG-3, GPG-113
 - 2D GPG-95
- link
 - animation GPG-269
 - animation engine GPG-280
 - animation engine and character GPG-272
 - array GPG-272
 - obtaining GPG-272
- link array GPG-283
 - getting GPG-283
 - setting GPG-283
- link data structure PPG-49
- link field DSG-26
- Link_Compute function GPG-283
- LinkAttachments() MPG-58
- linked list
 - adding a node to PPG-40
 - data structures PPG-46
 - traversing PPG-43
- linking 2D objects GPG-82
- linking DLLs PPG-295
- Links attribute GPG-268
- list data structure PPG-48

- list management
 - Portfolio GPG-94
- List nodes PPG-42
- ListAnchor PPG-48
- listen to AIFF file command TSD-56
- listen to AIFF File command TSD-56
- ListenerList data structure PPG-210
- listeners PPG-176
 - disconnection PPG-206
 - focus-dependent PPG-178
 - focus-independent PPG-178
 - focus-interested PPG-178
 - list of PPG-207
 - reconfiguring PPG-206
- literal color value GPG-149
- literal sample data MPG-46
- load instrument template MPG-20
- Load PIMap button TSD-21
- load rectangle GPG-159, GPG-160, GPG-162
 - defined GPG-162
 - described GPG-162
 - getting GPG-160
 - in runtime texture carving GPG-166
 - maximum size of GPG-162
 - obtaining GPG-167
 - setting GPG-160
- load rectangle attributes
 - setting GPG-167
- load sound file TSD-48
- LoadGameData() PPG-283, PPG-276, PPG-278
- LoadIcon() PPG-272, PPG-274
- loading TSD-48
 - dynamic PPG-288, PPG-289
- loading a PIMap file MPG-163
- loading the debugger DBG-8
- LoadInsTemplate MPG-26
- LoadInstrument() MPG-30
- LoadPIMap() MPG-160, MPG-161, MPG-163
- LoadSample() MPG-47
- LoadSoundFile() MPG-131
- Local GPG-246
- local axes GPG-242
- local character functions GPG-246
- local operation GPG-241
- Locale structure PPG-218
 - fields PPG-219
 - using PPG-223
- localization PPG-217
- Locations attribute GPG-136, GPG-140
- LockDisplay function GPG-224, GPG-225
- locking a resource PPG-79
- LockSemaphore() PPG-12, PPG-78, PPG-79
- LOD GPG-150, GPG-159
 - maximum number in a texture GPG-151
- LOD information
 - functions for setting and getting GPG-167
 - getting and setting GPG-164
- LODs
 - getting maximum number of GPG-163
- LogEvent() PPG-320
- LookAt GPG-247
- LookAt function GPG-246, GPG-133
- LookAt operation GPG-248
- LookupItem function GPG-202
- LookupItem() PPG-68, PPG-73, PPG-11
- LookupSymbol() PPG-299
- loop MPG-50, MPG-51, MPG-76
- looping sound effects TSD-5
- Lumberjack
 - creating PPG-320
 - deleting PPG-321
 - getting and releasing buffers PPG-321
 - getting information PPG-321
 - logging custom events PPG-320
 - parsing buffers PPG-322
 - starting and stopping PPG-320

M

- m PIMap flag TSD-19
- M2 APIs GPG-2
- M2 debugger
 - features of DBG-vii
- M2 debugger, see also debugger DBG-vii
- M2 development card, see also development (dev) card DBG-3
- M2 development system DBG-vii
- M2 FontBuilder FBR-4
- M2 FontBuilder dialog FBR-1
- M2 FontBuilder Preview window FBR-4
- M2 graphics
 - introducing GPG-1
- M2 graphics capabilities GPG-2
- M2 graphics hardware GPG-255, GPG-2
- M2 graphics library GPG-241

- M2 graphics system
 - animation techniques of GPG-xviii
 - APIs GPG-1
 - architecture of GPG-1
 - components of GPG-195
 - how it works GPG-256
 - rendering GPG-xviii
- M2 hardware
 - graphics GPG-255
- M2 hardware and software
 - responsibilities of GPG-xix
- M2 hardware components GPG-256
- M2 rendering engine GPG-1
- M2 rendering hardware GPG-152
- Macintosh DBG-viii, DBG-1, DBG-2, DBG-3
- Macintosh drag and drop, *see drag and drop*
- Macintosh operating system DBG-viii
- Macintosh Quadra DBG-viii
- Macintosh snd resource DSG-12
- Macintosh standard functionality PRO-14
- MacMix sound file TSD-54
- magnification filter CDM-25
- magnification region GPG-158
- main() function DBG-16, DBG-17, DBG-19, DBG-28
 - and DLLs PPG-288
- makefile DBG-4
 - newview DBG-12
- makepatch MPG-104
- MakeScore TSD-33
- MakeScore tool TSD-41
 - using to create 3SF file TSD-33
- malloc() PPG-5
- managing items PPG-67
- managing linked lists PPG-37
- managing memory PPG-319, PPG-51
- managing objects MPG-186
- manipulating characters GPG-241
- manipulating models GPG-xviii
- mapping
 - PIP CDM-32, CDM-33, GPG-151, GPG-153
 - texture CDM-19, GPG-145
- markers
 - decision functions MPG-141
 - in sounds MPG-138
 - static actions MPG-140
- markertime DSG-35
- mask
 - alpha GPG-192
- master copy VID-14
- mastering TSD-76
- Mastering software CDM-4
- MAT_Ambient attribute GPG-128
- MAT_Diffuse attribute GPG-128
- MAT_Emissive attribute GPG-128
- MAT_Specular attribute GPG-128
- MAT_TwoSided attribute GPG-128
- Material
 - diffuse color GPG-259
 - emissive color GPG-259
 - specular reflection GPG-259
- material GPG-256, GPG-127
 - ambient light GPG-259
 - characterizing GPG-258
 - emissive part GPG-125
 - matching with texture GPG-256
- material array GPG-247
- Material attribute GPG-112
- material emission GPG-125
- material functions GPG-128
- Material object GPG-128
- material object GPG-233
- Material objects GPG-107
- material properties GPG-258, GPG-260
 - listed GPG-260
- material properties (in SDF files)
 - defining GPG-242
- material properties (of shapes)
 - defining GPG-242
- Material, *see also* MatProp GPG-107
- materials GPG-232, GPG-3, GPG-109
 - and SDF files GPG-229
 - arrays of GPG-229
 - binding to surface GPG-293
 - setting GPG-258
- materials array
 - obtaining GPG-257
- Materials attribute GPG-257
- materials in SDF files GPG-242
- matrices
 - multiplying GPG-131
- MatProp functions GPG-128
- MatProp objects GPG-107
- MatProp struct GPG-128
- MatProp structure GPG-260
- MatProp structure functions GPG-128
- MatProp, *see also* Material GPG-260
- MatProp, *see also* Material> GPG-107

- matrices
 - adding GPG-131
 - post-multiplying GPG-133
 - transforming characters with GPG-249
 - transposing GPG-131
- matrix GPG-249
 - changing GPG-130
 - concatenating GPG-132, GPG-133, GPG-135
 - concatenating character matrix GPG-292
 - copying GPG-131
 - deleting GPG-129
 - initializing from a data array GPG-129
 - model/view GPG-249, GPG-292, GPG-123, GPG-135
 - model/view transformation GPG-135
 - model/view, see also model/view matrix GPG-135
 - post-multiplying GPG-131
 - replacing GPG-129, GPG-131
 - rotating GPG-131, GPG-132
 - scaling GPG-132
 - subtracting GPG-131
 - Transform GPG-135
 - transformation GPG-249, GPG-252, GPG-254, GPG-119, GPG-248
- matrix data
 - obtaining GPG-129
- matrix data array GPG-129
- matrix multiplication GPG-120
- matrix transformation GPG-292
- matrix transformations GPG-239
- matrix-creation operations GPG-129
- MatrixData typedef GPG-129
- MaxVoices MPG-160
- MDEV TSD-99
- MDFL TSD-99
- mechanism
 - extension GPG-233
- megabytes variable CDM-18
- MEMC_GIVE PPG-57, PPG-58
- MEMC_NOWRITE PPG-57
- MEMC_OKWRITE PPG-57
- MEMDEBUG
 - compile option PPG-59
- MemDebug PPG-59
- MEMDEBUGF_PAD_COOKIES option PPG-61
- MemInfo structure PPG-55
- memory
 - allocating PPG-5, PPG-53
 - buffer (for Bitmaps) GPG-202
 - flags PPG-53
 - getting from system-wide free memory pool PPG-58
 - internal GPG-256
 - organization of PPG-52
 - sharing PPG-9
 - system size PPG-5
- memory allocation PPG-5
 - cause failure PPG-61
 - checking for corruption PPG-60
- memory allocations
 - checking PPG-60
 - controlling PPG-230
- Memory Allocation Flags PPG-53
- memory block
 - continuous allocation of PPG-8
 - indexing PPG-54
- Memory Dump menu item DBG-60, DBG-68
- Memory Dump window DBG-40
- memory fence PPG-8
- memory information PPG-55
 - how big a block is PPG-55
 - how much avail PPG-55
- memory management PPG-2, PPG-51-PPG-62
- memory organization PPG-52
- memory page
 - getting size of PPG-56
- memory pages PPG-52
 - owning and sharing PPG-9
- memory pointers
 - validating PPG-56
- MEMTYPE_FILL PPG-53, PPG-55
- MEMTYPE_NORMAL PPG-53, PPG-55, PPG-56
- MEMTYPE_TRACKSIZE PPG-53, PPG-54, PPG-55
- menu commands DBG-51, DBG-61
 - Execute | Step In DBG-29
 - Execute | Step Over DBG-29
 - View | Breakpoints DBG-29
 - View | FP Registers DBG-45
 - View | PPC Supervisor Registers DBG-47
 - View | PPC Users DBG-44
 - View | Stack Crawl DBG-47
 - View | Task DBG-43, DBG-44, DBG-45, DBG-47
 - View | Triangle Disassembly DBG-49

- View | Variables DBG-39
- menu items DBG-51-??, DBG-61-??
 - Target | Memory Dump DBG-42
 - Target | Setup DBG-7
 - View | Send Variable Data DBG-36
 - View | Status DBG-17
 - View | Terminal DBG-9
 - View | Variables DBG-32
- menu reference DBG-51, DBG-61
- menus DBG-51, DBG-61
 - File DBG-10
- mesh
 - quadrilateral GPG-116
- message MPG-184
 - creating PPG-73
- message header DSG-26
 - fields DSG-26
- message items DSG-24
- message macros MPG-198
- message port DSG-24
 - creating PPG-73
 - example for creating DSG-25
- message ports PPG-14
 - creating PPG-15, PPG-88
- message queues PPG-15, PPG-88, PPG-92
- Message structure PPG-89, PPG-93
- messages DSG-25, PPG-178, PPG-14, PPG-83, PPG-87
 - buffered PPG-89, PPG-90, PPG-91, PPG-94, PPG-95
 - checking for PPG-92
 - configuration PPG-183, PPG-189
 - creating PPG-89
 - data block of PPG-91
 - example code PPG-96
 - for collections MPG-201
 - for sequences MPG-198
 - forwarding PPG-96
 - interpreting PPG-15
 - passing PPG-87
 - priority PPG-88
 - pulling back PPG-94
 - receiving PPG-15, PPG-92
 - replying to PPG-15, PPG-94
 - sending PPG-15, PPG-91
 - short PPG-89, PPG-90, PPG-95
 - small PPG-91, PPG-93
 - standard PPG-89, PPG-91, PPG-93, PPG-95
 - waiting for PPG-92
 - with no reply ports PPG-95
 - working with PPG-93
- message ports
 - finding PPG-96
- method MPG-184
 - calling directly MPG-198
- method macros MPG-210, MPG-211
- metronome timing PPG-120, PPG-122
 - vblank PPG-122
- MFDefineCollection() MPG-166
- MFLoadCollection() MPG-155, MPG-166
- MFLoadSequence() MPG-155, MPG-165
- MFLoadCollection() MPG-179
- microsecond clock PPG-117
- Microsoft WAVE sound file TSD-54
- MIDI MPG-149
 - channels MPG-151
 - Environment Calls MPG-181
 - example files TSD-18
 - file TSD-18
 - function calls MPG-181
 - importing a score MPG-155
 - messages MPG-151
 - playback calls MPG-182
 - preparation TSD-18
 - program number TSD-18
 - program numbers MPG-160
 - project file TSD-18
 - project interface tips TSD-23
 - providing playback functions MPG-156
 - review of MPG-151
 - Score Calls MPG-182
 - setting channels MPG-154
 - tricks TSD-24
 - working with real-time TSD-23
 - working with real-time MIDI TSD-23
- MIDI context format TSD-104
- MIDI environment
 - creating MPG-153
- MIDI event format TSD-105
- MIDI file MPG-4, TSD-31
 - playing on 3DO station TSD-20
- MIDI File button TSD-21
- MIDI files
 - loading and playing TSD-21
- MIDI folder TSD-13
- MIDI instruments MPG-90
- MIDI messages MPG-9
- MIDI network MPG-151

-
- MIDI Project window
 - importing PIMap TSD-20
 - MIDI score
 - overview of playing process MPG-157
 - playing MPG-168
 - setting voice and program limits MPG-158
 - MIDI score formats MPG-152
 - MIDI score playback
 - initializing a mixer for MPG-164
 - MIDI scores TSD-6
 - MIDI timing clocks MPG-152
 - MIDI tuning system MPG-90
 - MIDIFileParser MPG-165
 - Min/Max pop-up menu FBR-2
 - minfo_SysFree PPG-56
 - minfo_SysLargest PPG-56
 - minfo_TaskFree PPG-56
 - minification filter CDM-25
 - minification region GPG-158
 - MinimizeFileSystem() PPG-153
 - MinNode structure PPG-47, PPG-48
 - mipmap
 - described CDM-20, GPG-154
 - sizes of CDM-20, GPG-154
 - mipmap filtering CDM-21, GPG-154
 - tradeoffs in GPG-156
 - mipmap filtering modes CDM-21, GPG-147, GPG-155
 - mipmap texture filtering CDM-19, GPG-146
 - mipmapping CDM-19, GPG-146
 - mipmaps CDM-19, GPG-146
 - mixers MPG-20, MPG-22
 - MkNodeID() PPG-69, PPG-73
 - MkNodeID() macro PPG-69
 - Mod_Create function GPG-257
 - Mod_GetMaterials function GPG-257
 - Mod_GetTextures function GPG-258
 - Mod_SetMaterials function GPG-258
 - Mod_SetSurface function GPG-257
 - Mod_SetTextures function GPG-258
 - Mod_xxx functions GPG-289
 - model GPG-252
 - 3D GPG-xviii
 - animating GPG-280
 - changing texture GPG-258
 - character described geometrically GPG-262
 - converting to triangles GPG-256
 - creating GPG-257
 - defined GPG-262
 - geometric description of GPG-263
 - manipulating GPG-xviii
 - obtaining GPG-256
 - replacing surface GPG-257
 - SDF GPG-249
 - source of GPG-256
 - Surface object GPG-256
 - Model attributes
 - listed GPG-257
 - model class GPG-234
 - Model object
 - architecture of GPG-256
 - Model type GPG-289
 - model/view matrices
 - stack of GPG-135
 - model/view matrix GPG-249, GPG-292, GPG-123, GPG-135
 - model/view transformation matrix GPG-135
 - modeling functions
 - listed GPG-257
 - modeling programs GPG-256
 - models GPG-229, GPG-230, GPG-256, GPG-260, GPG-248
 - defined GPG-256
 - geometric GPG-238
 - ModelView attribute GPG-112
 - modes
 - filtering (for mipmaps) CDM-21, GPG-155
 - ModifyGraphicsItem function GPG-197, GPG-198, GPG-199, GPG-205, GPG-207, GPG-210, GPG-213, GPG-215, GPG-217, GPG-218, GPG-223
 - ModifyGraphicsItem() CDM-15
 - modifying
 - header file markers TSD-51
 - loop in sound file TSD-51
 - MIDI file header TSD-51
 - ModifyStorageReq() PPG-286
 - modular analog synthesizer MPG-19
 - module symbols PPG-289
 - modules
 - DLL PPG-288
 - functions for handling PPG-296
 - library PPG-288
 - monitor DBG-2, DBG-3
 - Monospace Characters checkbox FBR-3
 - motion
 - simulating GPG-xviii
 - MountFileSystem() PPG-153
-

- mouse PPG-102
 - monitoring PPG-215
 - return data PPG-196
- MouseEventData data structure PPG-196
- Move GPG-246
- Move operation GPG-238, GPG-248
- MovieCompress DSG-12, VID-24, VID-53, VID-28, VID-45
 - Color Quality Selection pop-up VID-46
 - functionality VID-45
- MovieCompress compression options
 - Animation compressor VID-45
 - Cinepak compression VID-46
- MovieCompress options
 - Frames per second pop-up VID-46
 - Quality slider for choosing compression quality VID-46
- MovieEdit DSG-12, VID-8, VID-28, VID-43
 - dynamic information VID-43
 - limitations VID-45
 - saving cross-platform movie VID-44
 - saving dependent movie VID-43
 - saving self-contained movie VID-44
 - window VID-43
- MovieEdit functionality
 - increase/decrease volume VID-44
 - jump to beginning VID-44
 - jump to end VID-44
 - loop VID-44
 - manual control speed VID-44
 - play forward VID-44
 - play in reverse VID-44
 - start playing movie VID-44
 - step frame forward VID-44
 - step frame reverse VID-44
 - stop playing movie VID-44
 - turn sound on or off VID-44
- MovieToStream DSG-13
 - where to find VID-24
- moving operation GPG-240
- MPEG DSG-44, DSG-46, DSG-47, VID-21, VID-23, VID-3, VID-5, VID-27
- MPEG audio
 - audio access unit DSG-40
 - audio frame DSG-40
 - audio presentation unit DSG-40
- MPEG Encoding dialog VID-32
- MPEG video
 - B-frame DSG-40
 - I-frame DSG-40
 - P-frame DSG-40
 - video access unit DSG-40
 - video presentation unit DSG-40
- MPEG-1 Video Elementary Stream VID-4
- MPEG-1 Video encoders VID-4
- MPEG-ability
 - Bitmap item GPG-200
- MPEGAudioChunkifier DSG-13, VID-6
- MPEGChunkifier VID-35, VID-36
- MPEGSplit VID-6
- MPEGVideoChunkifier DSG-13, VID-6, VID-8, VID-9
- MPW
 - and Portfolio Terminal window DBG-9
- MPW commands DBG-9
- MPW shell commands DBG-9
- MPW Worksheet window DBG-9
- msgItem field DSG-26
- Multibyte numbers GSG-50
- multim im parvo CDM-20, GPG-147
- multi-module applications PPG-289
- multiple inheritance GPG-295
- multiple parents
 - in character hierarchy GPG-261
- multiple-character operations GPG-249
- multiplication
 - matrix GPG-120
- multiplication values
 - getting and setting GPG-171
- multiplying spectra TSD-58
- multisampling MPG-53
- multitasking DSG-54, GPG-210, PPG-2
- music
 - creating TSD-5
 - looping TSD-5
- music for 3DO titles TSD-1
- Music library MPG-151, MPG-15
- musical score
 - playing MPG-4
- Mutation command TSD-61

N

-
- n_Name field PPG-47
 - n_Priority field PPG-39
 - name
 - symbolic GPG-232
 - symbolic, see symbolic name GPG-230
 - name index TSD-102
 - named object GPG-232
 - NamelessNode structure PPG-47
 - names
 - symbolic GPG-294, GPG-229
 - symbolic, see also symbolic names GPG-229
 - names of GPG-81
 - ne GPG-82
 - Nearest (point) filtering CDM-21
 - nearest (point) filtering CDM-24, CDM-26, GPG-147, GPG-155, GPG-157
 - negative Z-axis GPG-272
 - nesting
 - in character hierarchies GPG-261
 - New menu item DBG-52, DBG-62
 - New MIDI Project TSD-20
 - NewDataAcq() DSG-27
 - NewDataStream() DSG-27
 - newview application DBG-26
 - newview executable DBG-10, DBG-15
 - newview file DBG-4
 - newview folder DBG-4
 - newview makefile DBG-12
 - newview program DBG-3, DBG-14
 - setting up debugger for DBG-10
 - newview.c DBG-3, DBG-4, DBG-12, DBG-14, DBG-26
 - newview.make DBG-4
 - NeXT sound file TSD-54
 - NextNode() PPG-44
 - NextTagArg() PPG-35
 - node PPG-39
 - adding a node to the tail of a list PPG-40, PPG-41
 - adding according to alphabetical order PPG-41
 - adding according to other node values PPG-42
 - adding according to priority PPG-41
 - adding to a list PPG-40
 - adding to the head of a list PPG-40
 - changing priority of PPG-42
 - characteristics of PPG-39
 - finding by name PPG-45, PPG-46
 - of sprite object GPG-87
 - priority of PPG-39
 - removing from a list PPG-42
 - node priority PPG-41
 - node structure PPG-46
 - of sprite object GPG-87
 - node values PPG-42
 - node-structure attribute
 - of sprite object GPG-87
 - noise
 - compression considerations VID-15
 - Non-atomic chunk type GSG-46
 - non-UI events PPG-185
 - normals
 - generating GPG-139
 - transforming GPG-139
 - Normals attribute GPG-136, GPG-140
 - note bounce TSD-14
 - Note Off MPG-152, MPG-167
 - Note On MPG-167
 - NoteOffIns() MPG-174
 - NoteOnIns() MPG-174
 - notes
 - amplitude MPG-15
 - envelopes MPG-14
 - frequency MPG-15
 - pitch MPG-15
 - playing MPG-13
 - release MPG-14
 - sections of MPG-13
 - start MPG-14
 - stop MPG-14
 - voices MPG-15
 - NotesPerOctave value MPG-91
 - NoteTracker MPG-154, MPG-159, PPG-39
 - Notification by message PPG-106
 - Notification by signal PPG-106
 - NTSC DBG-7, GPG-256, PPG-120
 - NTSC basics GPG-257
 - NTSC coordinate standard GPG-204
 - NTSC hot colors DSG-10
 - NTSC scan lines GPG-259
 - NumericSpec structure PPG-220
 - numsubsmessages DSG-15
 - NuVista card VID-16

O

- Obj_Assign function GPG-286, GPG-289
- Obj_Delete function GPG-289
- Obj_FuncConstruct function GPG-290
- Obj_FuncCopy function GPG-291
- Obj_FuncDestroy function GPG-291
- Obj_FuncPrint function GPG-291
- Obj_GetType function GPG-111
- Obj_GetUse function GPG-289
- Obj_NewClass function GPG-289
- ObjArray
 - in SDF files GPG-294
- ObjArray type GPG-257
- object MPG-184
 - attributes of GPG-230
 - checking validity of MPG-188
 - copying GPG-291
 - creating MPG-187
 - creating destructor for GPG-291
 - defining constructor for GPG-290
 - defining copying procedure for GPG-291
 - deleting GPG-117
 - destroying MPG-187
 - engine GPG-281
 - establishing reference for GPG-289
 - example of GPG-232
 - getting reference count for GPG-289
 - getting reference count of GPG-289
 - in SDF file, see object GPG-230
 - in SDF files GPG-232
 - initializing GPG-290
 - material GPG-233
 - named GPG-232
 - printing attributes of GPG-291
 - rendering GPG-269
 - sending message to MPG-198
 - sprite, see also sprite object GPG-82
 - syntax for defining GPG-230
 - textured CDM-19, GPG-146
 - unnamed GPG-232
- object array GPG-283
- object defining tag arguments MPG-211
- object hierarchy GPG-229
- object list MPG-190
- object management calls MPG-211
- object type
 - getting GPG-111
 - SDF GPG-232
- object-oriented programming MPG-184
 - Graphics Framework and GPG-229
- objects GPG-3
 - 2D GPG-80
 - cel GPG-86
 - creating GPG-109
 - GP GPG-109
 - Graphics Framework GPG-228, GPG-231
 - illuminated GPG-124
 - properties of GPG-229
 - relationships of GPG-229
 - TexBlend GPG-229
 - viewable GPG-230
- objects (in SDF files)
 - attributes of GPG-229
- Objects folder DBG-5
- ObtainLumberjackBuffer() PPG-321
- of Graphics Pipeline GPG-109
- offset
 - PIP CDM-33, GPG-152
- OOP, see also object-oriented programming GPG-229
- opacity
 - of a texture GPG-151
- Open Any command TSD-55
- Open command PRO-14, TSD-54
- Open menu item DBG-62
- OpenAudioFolio() MPG-158
- OpenCompressionFolio() PPG-228
- OpenDeviceStack() PPG-105
- OpenDirectoryItem() PPG-152
- OpenDirectoryPath() PPG-152
- OpenFile() PPG-138, PPG-140, PPG-141
- OpenFileInDir() PPG-140, PPG-142
- OpenGraphicsFolio function GPG-81, GPG-82
- opening a device PPG-105
- opening any file as soundfile TSD-55
- opening headerless and text sound files TSD-55
- opening sound file
 - extensions TSD-54
 - types TSD-54
- OpenItem() PPG-75, PPG-11
- OpenModule() PPG-296, PPG-20
- OPENMODULE_FOR_TASK PPG-20
- OPENMODULE_FOR_THREA PPG-20
- OpenNamedDevice() PPG-116

OpenRawFile() PPG-146
 OpenSoundFile() MPG-131
 operating system
 Macintosh DBG-viii
 operating system components GPG-1
 operation
 global GPG-241
 local GPG-241
 operations
 geometric GPG-240, GPG-241
 multiple-character GPG-249
 on characters GPG-243
 on objects GPG-229
 optimizing compilation GPG-224
 optimizing stream files DSG-52
 optimizing streaming DSG-52
 Optional chunk GSG-49
 or GPG-143
 order-sensitive transformations GPG-249
 OrderViews function GPG-216, GPG-217, GPG-223, GPG-225
 and double-buffering GPG-225
 Orient operation GPG-248
 orientation field GPG-272
 orienting a character GPG-240
 origin
 camera and GPG-272
 OSType TSD-91
 Other menu FBR-5
 output buffer GPG-192
 output mixer
 specifying MPG-131
 Output Waveform dialog TSD-66
 Overlap option TSD-65
 override
 character GPG-292
 overrides
 engine GPG-293
 surface GPG-293
 overriding GPG-290
 overriding base-class functions GPG-290
 overriding functions GPG-289
 overstrike prevention GPG-96
 and 2D rendering GPG-96
 overview TSD-35

P

-p PIMap flag TSD-19
 packets DSG-41
 page boundaries GPG-201
 page phasing GPG-201
 Page Setup menu item DBG-63
 pages
 accessing GPG-201
 PAL DBG-7, GPG-256, PPG-120
 PAL video standard GPG-260
 palette index table, see also PIP table CDM-32, GPG-149, GPG-151
 palette lookup table, see also PIP table CDM-32, GPG-149
 panning MPG-118
 delay MPG-118
 parallel light rays GPG-276
 parameters
 Bitmap GPG-201
 parent
 in character hierarchy GPG-261
 parent character GPG-261
 parent class GPG-230
 parent tasks PPG-4, PPG-19
 parents
 multiple GPG-261
 ParseIFF() PPG-252, PPG-256, PPG-262
 parsing
 in SDF files GPG-230
 parsing bitfields GPG-230
 parsing files GPG-233
 parsing SDF files GPG-294, GPG-295
 Patch MPG-103
 PatchBay TSD-13, TSD-14
 PatchCmd MPG-104
 PatchDemo TSD-76
 path
 Data Directory DBG-63
 source directory DBG-63
 pathname PPG-140
 appending PPG-169
 determining PPG-169
 finding final component PPG-169
 specifications for PPG-140
 Pathnames CDM-7
 pathnames PPG-140
 Pause Process command TSD-70

- pausing a stream DSG-34
- PC indicator (->), see also program counter DBG-31
- PC VOX files MPG-214
- Pen Index Palette CDM-4
- PerformCopy() PPG-165, PPG-168
- perspective GPG-272
- perspective projection GPG-134
- perspective projection transformation GPG-134
- perspective view volume GPG-135
- phase increment MPG-72
- Phase Vocoder command TSD-64, TSD-65
- Phase Vocoder dialog TSD-64
- Phase Vocoder dialog options
 - Analyze Only TSD-65
 - Bands TSD-64
 - Filter window TSD-64
 - Overlap TSD-65
 - Pitch Scale TSD-65
 - Resynthesis Gating TSD-65
 - Scaling TSD-65
 - Scaling Function TSD-65
 - Time Scale TSD-65
- photo-optic gun PPG-102
- Photoshop VID-19, PRO-dcccxiii, PRO-2, PRO-5, PRO-6, PRO-10
- Photoshop to Texture dialog box PRO-10, PRO-11, PRO-15, PRO-16
- PIMap MPG-155
 - directly setting entries MPG-160
 - parts of TSD-18
 - setting entries MPG-160
 - using a MPG-161
 - working with TSD-18
- PIMap file MPG-4, TSD-18
 - example TSD-19
 - flags TSD-19
 - loading MPG-163
 - setting up multisamples MPG-162
- PIMP TSD-99
- PIP attributes
 - setting and getting GPG-170
- PIP mapping CDM-32, CDM-33, GPG-151, GPG-153
- PIP mapping stage CDM-33, GPG-153
- PIP offset CDM-33, GPG-152
- PIP RAM GPG-211
 - and TEContext item GPG-211
- PIP table CDM-32, GPG-144, GPG-149
 - and TEContext Item GPG-210
 - creating and managing GPG-154
 - defined GPG-151
 - described GPG-152
- PIP table of sprite object
 - sprite object
 - and PIP tables GPG-87
- PIP tables
 - creating and managing GPG-165
- PIP unit (of Triangle Engine) GPG-151
- Pip_Copy function GPG-165
- Pip_Create function GPG-154, GPG-165
- Pip_Delete function GPG-154, GPG-165
- Pip_GetData function GPG-165
- Pip_GetIndex function GPG-165
- Pip_GetSize function GPG-165
- Pip_SetData function GPG-165
- Pip_SetIndex function GPG-165
- Pip_SetSize function GPG-165
- PipColor struct GPG-154
- PipConst GPG-154
- pipeline
 - geometry GPG-135
- Pipeline, see also Graphics Pipeline GPG-227, GPG-105
- Pipelining objects
 - creating GPG-109
- PIP-table entry CDM-32, GPG-152
 - components of GPG-152
- PIP-Table information
 - setting and getting GPG-154
- Pitch GPG-247
- pitch MPG-90, MPG-109
 - changing MPG-46
- pitch bend MPG-93
- Pitch Bend Change MPG-152
- pitch bend change messages
 - acting on MPG-177
- pitch bend value
 - creating an internal MPG-178
- pitch bend wheel MPG-176
- pitch function GPG-244
- pitch operation GPG-246
- Pitch Scale Edit Function dialog TSD-66
- Pitch Scale option TSD-65
- pitch wheel MPG-93

- pixel
 - perspective correction CDM-6
 - texture coordinates CDM-6
 - x, y coordinates CDM-6
- pixel averaging mode
 - of Views GPG-206
- pixel blending, see also destination blending GPG-173
- pixel coordinates GPG-204
- pixel cornerweight MPG-xix
- Pixel lookup table chunk GSG-48
- pixel scaling CDM-61, GPG-173, GPG-175
- pixel size GPG-204
- pixel width
 - of Views GPG-207
- pixels GPG-3
- placeholders MPG-190
- Planar style RGB files GSG-47
- plane
 - clipping GPG-272, GPG-274
 - clipping, see also clipping plane GPG-135
 - front GPG-272
- planes
 - clipping GPG-271
- play TSD-31
- play 3DO score file TSD-37
- play back score file TSD-33
- play back with PlayScore TSD-33
- Play button TSD-21
- play current sound file TSD-49
- play in DSP TSD-31
- play in memory TSD-31
- play MIDI file TSD-31
- Play, Pause, Stop TSD-45
- playback MPG-35
 - changing characteristics during MPG-178
 - cleaning up after MPG-134
 - creating a MIDI score for MPG-180
 - examples DSG-17
 - tracking MPG-155
- playback from ARIA TSD-33
- playback functions MPG-156
- playback tempo MPG-168
- playing TSD-6
- playing a stream DSG-16
- playing MIDI scores TSD-6
- playing process MPG-157
- playing the MIDI score MPG-168
- PlaySA DSG-17
- PlayScore TSD-33, TSD-43
- PlayScore tool TSD-43
- PlayScore.lib TSD-43, TSD-44, TSD-45
 - ErrorProc TSD-44
 - ScoreReader TSD-43
- PLUT chunk GSG-48
- pod table PPG-199
 - gaining access to PPG-200
 - relinquishing PPG-200
 - synchronizing PPG-206
- PodData data structure PPG-211
- PodDescription data structure PPG-208
- PodDescriptionList data structure PPG-207
- Pods PPG-176
 - commanding PPG-211
 - listing connected PPG-207
- point
 - 2D GPG-121
 - 3D GPG-121
 - homogeneous GPG-121
 - initializing GPG-121
 - transforming GPG-122
- point at infinity GPG-121
- point functions GPG-121
- point light GPG-277
- point primitives
 - 2D GPG-95
- point set
 - drawing GPG-117
- point type
 - SDF GPG-232
- Point2 structure GPG-118
- Point3 structure GPG-118
- Point4 structure GPG-118
- points GPG-3, GPG-113, GPG-118
 - 2D GPG-95
 - 3D GPG-249
 - getting distance between GPG-122
 - operations on GPG-121
 - transformations on GPG-119
 - transforming GPG-122
- polar coordinates GPG-133
- polygons
 - defining GPG-241
 - drawing GPG-3
 - material properties and GPG-260
- pop GPG-111, GPG-135
- Pop and Push functions GPG-111
- PopChunk() PPG-254, PPG-255, PPG-269

- Portable Pixel Map GSG-58
- Portfolio devices PPG-115
- portfolio devices
 - communicating with PPG-116
- Portfolio list PPG-38
 - and 2D objects GPG-94
- Portfolio list management routines GPG-94
- Portfolio lists
 - and 2D objects GPG-94
- portfolio lists
 - characteristics of PPG-38
- Portfolio node
 - and sprite object GPG-87
- portfolio node
 - of sprite object GPG-83
- Portfolio Terminal DBG-57, DBG-66
- Portfolio Terminal window DBG-55, DBG-64
- Portfolio Terminal window DBG-27
 - and MPW DBG-9
 - described DBG-9
 - printing DBG-63
 - using as a help window DBG-9
- position
 - camera GPG-xviii
 - lighting GPG-xviii
 - of viewer GPG-133
- position field GPG-272
- positions
 - of sprite objects GPG-91
- postion
 - lighting GPG-xviii
- post-multiplying matrices GPG-131
- Power Macintosh DBG-viii
- Power PC FP Registers window DBG-45
- Power PC registers, see registers
- Power PC Supervisor Registers window DBG-46
- Power PC User Registers window DBG-44
 - using DBG-44
- PowerBus GPG-256
- PowerPC 602 GPG-256
- PowerPC 602 CPU GPG-256
- PPC GPG-256
- PPC FP (floating-point) Registers window DBG-43
- PPC FP Registers menu item DBG-57, DBG-66
- PPC Supervisor Registers menu item DBG-57
- PPC Supervisor Registers window DBG-43
- PPC User Registers menu item DBG-57, DBG-66
- PPC User Registers window DBG-43, DBG-44
- PPC Users Registers window DBG-44
- PPM GSG-58
- preface GPG-xvii
- preferences
 - debugger DBG-6
 - setting DBG-6
- Preferences dialog box DBG-54, DBG-64
- preferences file DBG-2
- preloading a sound file MPG-130
- preloadinstrument DSG-29, DSG-15
- pre-multiplying matrices GPG-131
- Preparing MPG-26
- preparing TSD-76
- preparing a stream file DSG-6
- preparing AIFF files TSD-4
- preparing film source material VID-14
- preparing for DBG-1
- Preparing optimized CD-ROM image CDM-16, CDM-17
- Preparing simple CD-ROM image CDM-13
- preparing stream files DSG-6
- PREPLIST macro PPG-40
- PrepList() PPG-39
- presentation timestamp DSG-43
- previously compiled surfaces
 - rendering of GPG-228
- previously constructed surfaces GPG-106
- PrevNode() PPG-45
- primitive
 - faceted GPG-247
 - finding GPG-144
- primitive color CDM-34, GPG-158
- primitive drawing attributes GPG-113
- primitives GPG-95
 - 2D GPG-3
 - geometry of GPG-118
 - GP GPG-113
 - rendering GPG-115
- printf statements DBG-60, DBG-68
- printf() PPG-311, PPG-313
- printing object attributes GPG-291
- PrintObject() MPG-201, MPG-202
- priorities DSG-54
 - problems DSG-56
- privatePtr field DSG-26
- Probes
 - reading MPG-89
- processing digitized materials
 - 3-2 Pulldown VID-18
 - deinterlacing VID-18

processing file with TSD-58
 processing paths
 diagram VID-5
 factors VID-3
 MPEG VID-5
 Program Change MPG-152
 program counter symbol (->) DBG-19
 program-counter indicator (->) DBG-28
 project management TSD-8
 Projection attribute GPG-112
 projection transformation
 perspective GPG-134
 prompt
 remote> DBG-9
 properties
 material GPG-258, GPG-260
 properties (of objects) GPG-229
 providing voices MPG-154
 pstring TSD-91
 Pt2_Set function GPG-121
 Pt3_Distance function GPG-119
 Pt3_Print function GPG-119, GPG-122
 Pt3_Set function GPG-119, GPG-121
 Pt3_Transform function GPG-119, GPG-122
 Pt4_Print function GPG-119, GPG-122
 Pt4_Set function GPG-119, GPG-121
 Pt4_Transform function GPG-119, GPG-122
 push GPG-111, GPG-135
 Push and Pop functions GPG-111
 PushChunk() PPG-254, PPG-255, PPG-268,
 PPG-269
 pyramid
 viewing GPG-134

Q

QMesh_Create function GPG-137, GPG-140
 QMesh_GetColor function GPG-140
 QMesh_GetLocation function GPG-140
 QMesh_GetNormal function GPG-140
 QMesh_GetTexCoord function GPG-140
 QMesh_Init function GPG-140
 QMesh_SetColor function GPG-141
 QMesh_SetLocation function GPG-141
 QMesh_SetNormal function GPG-141
 QMesh_SetTexCoord function GPG-141
 QTVideoChunkifier VID-53, VID-10, VID-28
 QuadMesh GPG-143
 initializing GPG-140

QuadMesh functions GPG-140
 QuadMesh Geometry struct GPG-137
 QuadMesh structure GPG-139, GPG-142
 architecture of GPG-139
 constructing GPG-140
 creating GPG-140
 defined GPG-139
 Quadra DBG-viii
 quadrilateral
 arbitrary GPG-86
 quadrilateral mesh
 drawing GPG-116
 quadrilateral mesh structure, see also QuadMesh
 GPG-139
 Quality option TSD-69
 quantum remaindering PPG-24
 quanta PPG-24
 Quasi tri-linear filtering CDM-21
 quasi tri-linear filtering CDM-26, GPG-147, GPG-
 156, GPG-157
 QueryStorageReq() PPG-283, PPG-285
 Quick I/O PPG-110
 QuickTime VID-24, VID-16, VID-51, VID-2, VID-
 5, VID-7, VID-9, VID-10, VID-27
 Animation component VID-7
 animation component VID-7
 QuickTime movie
 compression options VID-25
 edit with MovieEdit VID-43
 field dominance VID-39
 play with MovieEdit VID-43
 QuickTOPIX CDM-4, CDM-25
 setup CDM-24
 Quit command PRO-14, TSD-56

R

radius attribute GPG-229
 RAM-resident vs. ROM-resident sound TSD-2
 raster line GPG-214
 rasterization GPG-257
 rasterized image GPG-257
 RasterOps car VID-16
 Rate Control VID-30
 ratio
 aspect GPG-271, GPG-274
 RationMemDebug() PPG-60, PPG-61
 raw files TSD-49
 RawFile

- clearing errors PPG-149
- closing PPG-151
- getting info PPG-149
- opening PPG-146
- reading PPG-147
- seeking in PPG-148
- setting attributes PPG-150
- setting size PPG-148
- writing to PPG-147
- RawFile functions PPG-138, PPG-145
- read/write TSD-49
- read/write raw files TSD-49
- ReadBattClock() PPG-172
- ReadChunk() PPG-252, PPG-269
- ReadDirectory() PPG-153
- Reading MPG-40
- reading a collection's object list MPG-197
- reading data
 - from a file PPG-128
- Reading files from CD-ROM CDM-7
- ReadRawFile() PPG-147
- ready queue PPG-3
- ready to run
 - task state PPG-3
- Reallocating a Block of Memory PPG-54
- ReallocMem() PPG-54, PPG-55
- real-time MIDI TSD-23
 - working with TSD-23
- Receive Data DBG-34
- Recording speed CDM-24
- rectangle primitives
 - 2D GPG-95
- rectangles
 - 2D GPG-95
- Red Book to AIFF TSD-4
- red, green, blue, and alpha components CDM-6
- reference
 - character GPG-261
- reference count
 - getting GPG-289
- reference-counting GPG-261, GPG-283, GPG-289
- reflection
 - specular GPG-259
- regions
 - display GPG-214
 - viewable GPG-214
- Register Commands CDM-63
- register settings
 - and TEContext Item GPG-210
- RegisterCollectionChunks() PPG-251, PPG-258, PPG-263
- registering classes GPG-233
- RegisterPropChunks() PPG-263
- RegisterPropChunks() PPG-259, PPG-261
- registers DBG-vii
 - and TEContext item GPG-211
 - constant CDM-28, GPG-148
 - control GPG-211
 - modifying contents of DBG-44
 - viewing contents of DBG-44
- Registers User window
- Registers windows DBG-43
- RegisterStopChunks() PPG-252, PPG-262, PPG-263
- REIMPORT_ALLOWED flag PPG-292
- relative MPG-40
- Relative pathnames CDM-7
- relative pathnames PPG-140
- release loop MPG-51, MPG-76
- ReleaseAttachment() MPG-57
- ReleaseInstrument() MPG-37
- ReleaseLumberjackBuffer() PPG-321
- ReleaseScoreNote() MPG-174
- releasing a note MPG-173
- releasing resources PPG-4
- relevant event times MPG-203
- RemHead() PPG-42
- RemNode() PPG-43
- remote> prompt DBG-9
- RemoveContextInfo() PPG-267
- RemoveNthFromObject() MPG-197, MPG-202
- RemoveView function GPG-217, GPG-223
- removing a breakpoint
 - BreakPoints window DBG-29
- removing a node PPG-42
- removing a specific node PPG-43
- removing composite frames
 - 3-2 Pulldown VID-37
- removing the last node PPG-43
- RemTail() PPG-43
- Rename() PPG-141
- render list
 - of 2D Graphics Framework objects GPG-82
- render signal GPG-217, GPG-219, GPG-220, GPG-222
- Render Signals GPG-217
- Renderability
 - Bitmap item GPG-200

- rendered images
 - making them visible GPG-1
- rendering GPG-228, GPG-111, GPG-xviii
 - 2D GPG-96
 - 3D GPG-2
 - and overstring prevention GPG-96
 - Bitmap GPG-219
 - bitmap for GPG-111
 - deferred GPG-143, GPG-228
 - of frames GPG-xviii
 - optimizing GPG-143
- rendering 2D objects GPG-81, GPG-94
- rendering a ghost CDM-33, GPG-153
- rendering attribute functions GPG-113
- rendering attributes
 - GP GPG-113
- rendering capabilities
 - 3D GPG-1
- rendering characters GPG-292
- rendering engine
 - M2 GPG-1
- rendering functions GPG-115
 - GP GPG-117
- rendering geometric primitives GPG-117
- rendering hardware GPG-3
 - M2 GPG-152
- rendering instructions GPG-210, GPG-211
- rendering library
 - 3D GPG-1
- rendering operations GPG-109
 - functions for GPG-115
 - GP GPG-113
- rendering primitive functions GPG-115
- rendering primitives GPG-115
 - GP GPG-113, GPG-114, GPG-115
- rendering scenes and objects GPG-269
- rendering sprites GPG-87
- rendering-control attributes GPG-117
- rendering-control functions GPG-117
- reorder_array function GPG-286
- reply ports PPG-15, PPG-94
- ReplyMsg() DSG-26, PPG-181, PPG-191, PPG-94, PPG-95, PPG-96
- ReplySmallMsg() PPG-15, PPG-95, PPG-96
- Requestor folio PPG-281
 - how to use PPG-282
 - purpose PPG-281
- requestor object
 - creating PPG-283
- Required chunk GSG-49
- requirements
 - system DBG-vii, DBG-viii
- Requirements for CD-ROM mastering CDM-1
- Reset GPG-247
- resizing
 - QuickTime movie VID-19
 - software for VID-19
- resolution
 - color GPG-260
 - display GPG-258
 - intensity GPG-260
- resource
 - locking PPG-79
 - unlocking PPG-80
- resource limitations DSG-8
- resource sharing PPG-2
- Resynthesis Gating option TSD-65
- returning memory PPG-9
- reverb
 - MPG-111, MPG-112, MPG-113
- reverberation MPG-94
 - delay instrument MPG-94
 - delay line MPG-94
- reverberations
 - attachment starting point MPG-98
 - complex MPG-99
 - simplest MPG-111
 - submixer levels MPG-98
- RGB GPG-151
- RGB components GPG-124
- RGBA color values
 - of extended sprite objects GPG-88
- right-hand world coordinate system GPG-133
- right-handed coordinate system GPG-241
- Ring Modulate option TSD-59
- Roll GPG-246
- roll function GPG-244
- roll operation GPG-246
- Rotate GPG-247
- Rotate operation GPG-238
- rotate transformation GPG-132
- Rotates GPG-246, GPG-247
- RotateScene() function DBG-28
- rotating GPG-240, GPG-248
- rotating a character GPG-246
- rotating characters GPG-240, GPG-247
- rotating function GPG-244
- rotating operation GPG-240

- rotating sprite objects GPG-83
- rotation GPG-247, GPG-249, GPG-263, GPG-129
- rotation functions
 - for characters GPG-247
- rotational angles GPG-241
- round-robin scheduling PPG-3, PPG-24
- row vector GPG-119
- run-length encoding GPG-148
- running
 - task state PPG-3
- runtime texture carving GPG-166
 - functions used in GPG-167

S

- sample TSD-31
 - loading MPG-20
- sample item MPG-47
- sample rate conversio VID-11
- sample rate for sound TSD-3
- sample size MPG-46
- sample sound pointers MPG-13
- sample,play in memory TSD-31
- sample,spool from disk TSD-31
- sample,weave into stream TSD-31
- sampled music TSD-5
- sampled sounds MPG-127
- sampled-sound instrument
 - raw value MPG-73
- sampled-sound instruments MPG-22, MPG-23, MPG-46
 - knob MPG-73
 - mono MPG-46
 - stereo MPG-46
- samples MPG-46
 - attaching multisamples to instruments MPG-53
 - attaching to instrument MPG-51
 - attachment MPG-53
 - debugging MPG-54
 - deleting MPG-54
 - detaching MPG-54
 - FIFO MPG-52
 - frame index MPG-50
 - loading MPG-46
 - loops MPG-50
 - multisampling MPG-53
 - playback MPG-51
 - release loop MPG-51
 - sample frame MPG-50
 - simplest loading method MPG-47
 - start point MPG-55
 - starting dependent MPG-34
 - sustain loop MPG-51
 - tag args MPG-48
 - trigger points MPG-51
- Samples folder TSD-13
- SampleSystemTimeTV() PPG-118
- SampleSystemTimeVBL() PPG-121
- sampling sound effects TSD-4
- SanityCheckMemDebug() PPG-60
- SAudio subscriber DSG-17
- Save A Copy command TSD-55, TSD-56
- Save anti-aliased font FBR-4
- Save menu item DBG-52, DBG-62
- Save PIP RAM
 - TEContext property GPG-211
- Save Registers
 - TEContext property GPG-211
- Save Texture RAM
 - TEContext property GPG-211
- SaveGame
 - FORM SGME chunk PPG-276
 - SGDATA structure PPG-278
- SaveGame folio PPG-283, PPG-275
 - functions PPG-276
 - purpose PPG-275
- SaveGameData() PPG-276
- SaveIcon() PPG-272, PPG-274
- saving fonts FBR-4
- saving in 3DO compressed format TSD-56
- Scale operation GPG-238, GPG-248
- scale transformation GPG-133
- scaling
 - of a character GPG-240
 - pixel GPG-173
 - pixel, see pixel scaling
- scaling amplitude of sound file TSD-61
- scaling characters GPG-249
- Scaling Function option TSD-65
- scaling gain TSD-61
- Scaling option TSD-65
- scaling sprite objects GPG-83
- scan lines GPG-258, GPG-259
- scan pattern GPG-259
- ScanList() PPG-43
- ScanListB PPG-44
- ScavengeMem() PPG-53, PPG-58

- scene
 - 3D GPG-2
 - background color GPG-271
 - camera and GPG-272
 - clearing GPG-271
 - components of GPG-229
 - creating GPG-269
 - defined GPG-248
 - deleting GPG-269
 - designating animation engines for GPG-271
 - dynamic characters in GPG-269
 - lighting of GPG-270
 - rendering GPG-269, GPG-3
 - searching GPG-269
 - setting visibility of GPG-270
 - static characters in GPG-269
- scene attributes GPG-269
- scene background GPG-269, GPG-271
- scene boundary GPG-272
- Scene class GPG-295
- scene class GPG-234
- Scene Description Format
 - bit-mask attribute GPG-232
 - enumerated attribute GPG-232
 - see also SDF GPG-232
 - shapes
 - iradius GPG-237
 - oradius GPG-237
 - verticesPerColumn GPG-241
 - verticesPerRow GPG-241
 - textures
 - texture coordinates GPG-243
- scene description format, see also SDF GPG-228
- Scene Description Format, see SDF GPG-3
- scene description format, see SDF GPG-227
- scene functions
 - functions for working with scenes GPG-269
- Scene object GPG-269
- scene transparency GPG-271
- scene visibility GPG-270
- Scene_Create function GPG-269
- Scene_Delete function GPG-269
- Scene_Display function GPG-269, GPG-282
- Scene_GetBackColor function GPG-271
- Scene_GetBound function GPG-270
- Scene_GetCamera function GPG-270
- Scene_GetDynamic function GPG-270
- Scene_GetEngines function GPG-271, GPG-282
- Scene_GetFrameOptions function GPG-271
- Scene_GetLight function GPG-270
- Scene_GetLinks function GPG-272, GPG-283
- Scene_GetStatic function GPG-270
- Scene_IsAutoAdjust function GPG-271
- Scene_IsTransparent function GPG-271
- Scene_IsVisible function GPG-270
- Scene_SetAutoAdjust function GPG-271
- Scene_SetBackColor function GPG-271
- Scene_SetBound function GPG-270
- Scene_SetDynamic function GPG-269
- Scene_SetEngines function GPG-271, GPG-282
- Scene_SetFrameOptions function GPG-271
- Scene_SetLinks function GPG-272, GPG-283
- Scene_SetTransparent function GPG-271
- Scene_SetVisibility function GPG-270
- scenes
 - 3D GPG-2
 - animating GPG-xvii
 - SDF GPG-248
- score context
 - creating MPG-159
- score file TSD-32, TSD-33
- score file playback TSD-33
- score item TSD-101
- score items TSD-96
- score playback MPG-152
- ScoreChannel MPG-154
- ScoreContext MPG-155, MPG-159, MPG-177
- scores
 - files MPG-216
 - playing MPG-215
- screen description format
 - defined GPG-2
- script
 - debugger DBG-6
- Script folio PPG-315
- script, see also debugger script DBG-7
- SCRIPT_TAG_BACKGROUND_MODE PPG-316
- SCRIPT_TAG_BACKGROUND_MODE tag PPG-317
- SCRIPT_TAG_CONTEXT tag PPG-316
- ScriptContext struct PPG-316
- ScriptContext structure PPG-316, PPG-317
- SDF GPG-228, GPG-231, GPG-237
 - defined GPG-231, GPG-2
 - described GPG-3
- SDF (scene description format) file DBG-14

- SDF array GPG-294, GPG-247
- SDF bitfields, see bitfields GPG-230
- SDF camera GPG-249
- SDF class descriptions GPG-263
- SDF classes GPG-294
 - and Graphics Framework classes GPG-230
 - and Graphics Pipeline classes GPG-230
 - restrictions in GPG-295
- SDF data fields GPG-295
- SDF dictionary GPG-232, GPG-273, GPG-229
 - searching GPG-257
- SDF enumerations, see enumerations GPG-230
- SDF examples GPG-249
- SDF file GPG-269
 - benefits of GPG-228
 - bit-mask attribute GPG-232
 - class definitions in GPG-294
 - classes of objects in GPG-229
 - defined GPG-227
 - defining a classes in GPG-294
 - described GPG-228
 - enumerated attribute GPG-232
 - features of GPG-228
 - formats of GPG-2
 - including other files in GPG-231
 - materials in GPG-242
 - parsing GPG-294, GPG-295
 - punctuation in GPG-231
 - quotation marks in GPG-232
 - strings in GPG-231
- SDF files GPG-294, GPG-297, GPG-227
 - ASCII GPG-227
 - ASCII-format GPG-231
 - binary GPG-227
 - binary and ASCII GPG-229
 - binary format GPG-231
 - comments in GPG-231
 - data types in GPG-231
 - formats of GPG-231
 - Graphics Framework extensions GPG-294
 - Graphics Framework extensions in GPG-294
 - how they work GPG-232, GPG-229
 - textures in GPG-242
 - writing GPG-231
- SDF library GPG-229
- SDF light GPG-249
- SDF model GPG-249
- SDF names GPG-294
- SDF scene
 - defined GPG-248
- SDF scenes GPG-248
- SDF surface GPG-245, GPG-248
- SDF symbolic names GPG-294
- SDF TexBlend class GPG-244
- SDF TexBlend objects GPG-242
- SDF types
 - built-in GPG-234
 - common GPG-234
 - listed GPG-234
- SDF, see also Scene Description Format GPG-241
- SDF_Close function GPG-234, GPG-289
- SDF_FindObj function GPG-257, GPG-269, GPG-273, GPG-289
- SDF_Open function GPG-233
- SDF_Parse function GPG-233
- SDF_Register function GPG-233, GPG-234
- se GPG-280
- sections
 - rendering 2D sprites in GPG-85
- see also M2 debugger DBG-vii
- SeekChunk() PPG-270
- SeekRawFile() PPG-148
- semaphore PPG-12
 - creating PPG-73
 - deleting PPG-80
 - finding PPG-80
 - using PPG-78
- semaphores PPG-65, PPG-77
- semitones TSD-66
- Send Variable Data menu item DBG-58, DBG-66
- SendIO PPG-110
- SendIO() PPG-116, PPG-109
- SendMsg() PPG-15, PPG-91
- SendSignal() PPG-14, PPG-86
- SendSmallMsg() PPG-91
- sequence
 - setting up event list for MPG-191
 - setting variables with tag argument values MPG-193
 - special-effects GPG-xvii
- sequence class MPG-188, MPG-189
 - event list MPG-189
- sequencer TSD-14
- SER_CMD_BREAK PPG-135
- SER_CMD_GETCONFIG PPG-134
- SER_CMD_SETCONFIG PPG-133
- SER_CMD_SETDTR PPG-135
- SER_CMD_SETRTS PPG-135

- SER_CMD_STATUS PPG-135
- SER_CMD_WAITEVENT PPG-134
- SerConfig structure PPG-133
- serial devices PPG-115
- serial lines
 - controlling PPG-135
 - DTR PPG-135
 - RTS PPG-135
- serial port TSD-14
 - configuring PPG-133
 - getting state of PPG-135
 - reading configuration PPG-134
 - reading data from PPG-132
 - writing data to PPG-131
- service function MPG-137, MPG-144
- ServiceSoundFile() MPG-132
- set header TSD-99
- Set Leading and Spacing dialog FBR-5
- SetAudioItemInfo() MPG-88
- SetAudioRate() MPG-179
- SetBg command PPG-316
- SetBitRange() PPG-64
- SetFg command PPG-316
- SetFileAttrs() PPG-142
- SetFocus data structure PPG-211
- SetItemOwner PPG-4
- SetItemOwner() PPG-74, PPG-23
- SetItemPri() PPG-74, PPG-11, PPG-23
- SetMaxSize function GPG-285
- SetNodePri() PPG-39, PPG-42, PPG-48
- SetObjectInfo() MPG-195, MPG-196, MPG-199
- SetPIMapEntry() MPG-160, MPG-161
- SetRawFileAttrs() PPG-150
- SetRawFileSize() PPG-148
- SetScoreBendRange() MPG-176, MPG-177
- setting a breakpoint
 - Breakpoints window DBG-29
- setting a pitch bend range value MPG-176
- setting background color GPG-271
- setting channel panning and volume MPG-175
- setting channels MPG-154
- setting data rate for Cinepak compression VID-25
- setting header channel number TSD-50
- setting header data format TSD-50
- setting header sample rate TSD-50
- setting PC
 - Disassembly window DBG-21
- setting up a MIDI score MPG-158
- setting up a mixer MPG-164
- setting up multisamples MPG-162
- setting up objects for a collection MPG-195
- Setup menu item DBG-59, DBG-68
- Setup submenu DBG-53, DBG-63
- sfStartReading TSD-44
- SFToStream DSG-13
- SGME chunk PPG-276
- Shade Enable field GPG-242
- Shade Enable material GPG-114
- ShadeEnable field
 - in MatProp struct GPG-128
- shading GPG-106
- shading effects
 - on extended sprite objects GPG-88
- shape
 - material properties of GPG-242
- shared resources
 - SEE ALSO items PPG-10
- sharing memory PPG-9
- sharing system resources PPG-77
- shell commands
 - MPW DBG-9
- shift CDM-7
- Shine material GPG-114
- shininess attribute GPG-259
- shininess parameter GPG-242
- shooting video VID-14
- short sprite object GPG-81
- short type
 - SDF GPG-232
- Show Output command TSD-70
- Show Spectrum command TSD-70
- shuffle_textures example function GPG-258
- SIGF_IODONE PPG-106
- sight
 - line of GPG-133, GPG-134
- signal
 - dispatching GPG-217, GPG-223
 - display GPG-217
 - display, see also display signal GPG-219
 - render GPG-217
 - render, see also render signal GPG-219
 - requesting GPG-217
 - View GPG-217
- signal bits
 - allocation PPG-84
 - freeing PPG-86
 - sampling, changing PPG-86
- signal masks PPG-13

- signaling
 - View GPG-207
- Signals
 - Display GPG-217
 - Render GPG-217
- signals PPG-13, PPG-83
 - blocking GPG-223
 - freeing PPG-14
 - receiving PPG-14, PPG-85
 - sample code PPG-87
 - sending PPG-14, PPG-86
 - used with messages PPG-88
 - using PPG-84
 - view GPG-217
 - waiting for PPG-85
- single inheritance GPG-295
- Size GPG-246
- size
 - obtaining GPG-264
- Size attribute GPG-136, GPG-140
- Size limitations for CD-ROM mastering CDM-6
- size of pixels GPG-204
- sizing operation GPG-240
- sleep PPG-14
- slicing a sprite object GPG-86
- SMPTE TSD-6
- SNDS chunk DSG-7
- soft spotlight GPG-279
- software
 - 3DO GPG-3
- Software requirements for CD-ROM mastering CDM-3
- Some DSG-39, DSG-41
- SOPT_FLUSH DSG-34
- SOPT_NOFLUSH DSG-34
- sound
 - compressed TSD-3
 - converting DSG-12
 - decompression TSD-3
 - directionality MPG-119
 - environment MPG-120
 - fading smoothly MPG-6
 - harmonic relations TSD-7
 - looping TSD-5
 - producing MPG-2
 - project management TSD-8
 - RAM-resident vs. ROM-resident TSD-2
 - sample rate TSD-3
 - spooled vs. streamed TSD-2
 - stereo vs. mono TSD-3
 - synchronization TSD-8
 - volume settings TSD-7
- sound design TSD-7
- sound design with TSD-32
- sound effects
 - complex MPG-6
 - for 3DO titles TSD-1
 - managing MPG-5
 - sampling TSD-4
- sound file TSD-48
 - loading TSD-48
 - modify loop TSD-51
 - playing MPG-3, TSD-48
 - playing any number of times MPG-136
 - playing once MPG-135
 - playing repeatedly MPG-135
 - preloading MPG-130
 - scaling amplitude TSD-61
 - servicing MPG-132
- sound file playback
 - starting MPG-131
- sound file player
 - creating MPG-129
 - creating buffers for MPG-130
 - rewinding MPG-133
 - typical calling sequences MPG-135
- sound files
 - loading multiple MPG-215
- sound set TSD-95, TSD-101
- sound spooler MPG-141
 - buffers MPG-144
 - convenience routines MPG-147
 - example MPG-143
 - function calls MPG-148
 - how it works MPG-142
 - how to use MPG-142
 - operating MPG-129
 - operation MPG-128
 - overview of MPG-128
 - starvation MPG-144
- Soundfile Information window TSD-48, TSD-54
- soundfile playback
 - stopping MPG-133
- SoundFilePlayer data structure MPG-129
- SoundHack VID-11, TSD-75, TSD-48, TSD-54, TSD-56, TSD-70
- SoundHack Control menu TSD-70
- SoundHack File menu TSD-54, TSD-55, TSD-56

-
- SoundHack Hack menu TSD-57, TSD-58, TSD-61, TSD-64, TSD-66, TSD-68
 - sound-synthesis instrument
 - starting MPG-34
 - sound-synthesis instruments MPG-22, MPG-23
 - Source Directory path DBG-63
 - source files DBG-10
 - source frame buffer GPG-193
 - blending with GPG-193, GPG-194
 - source image specification GPG-175
 - source material
 - master copy VID-14
 - preparing VID-14
 - shooting video VID-14
 - typical VID-14
 - Source menu item DBG-56, DBG-66
 - source selection bit, see also SSB CDM-28, GPG-148
 - Source window DBG-18, DBG-19, DBG-28, DBG-56, DBG-58, DBG-66
 - introduced DBG-18
 - opening DBG-19
 - source-line symbol (#) DBG-19
 - sources
 - of light GPG-123
 - sources of lights GPG-275
 - space CDM-13
 - SPAction MPG-142
 - Sparkle VID-21, VID-4, VID-8, VID-27, VID-28
 - spawning tasks PPG-5
 - spBranchatMarker() MPG-139
 - spDeletePlayer() MPG-138
 - Special Mode DBG-54, DBG-63
 - Special mode DBG-53, DBG-63
 - Special Mode menu item DBG-53
 - special tag commands PPG-32
 - special-effects sequence GPG-xvii
 - specifying TSD-101
 - specifying a user context MPG-166
 - specifying score item TSD-101
 - specifying sound set TSD-101
 - Spectral Dynamics command TSD-66
 - Spectral Dynamics Processor dialog TSD-67
 - Bands pop-up TSD-68
 - Gain/Reduction TSD-68
 - Gate-Duck pop-up TSD-67
 - Highest/Lowest Band TSD-68
 - Threshold Level TSD-68
 - Spectral Mutation dialog TSD-61
 - specular color GPG-259
 - specular component GPG-127
 - specular component of GPG-127
 - Specular flag GPG-242
 - specular light GPG-125
 - Specular material GPG-114
 - Specular property GPG-260
 - specular reflection GPG-259
 - Speed attribute GPG-281
 - Sphere class GPG-229
 - Spherical type GPG-243
 - spite object
 - rendering GPG-87
 - splash screen PRO-4
 - spLinkSounds() MPG-140
 - spLoopSound() MPG-140
 - SPMarker MPG-138
 - spool from disk TSD-31
 - Spoiled vs. streamed sound TSD-2
 - spotlight GPG-277
 - angle of GPG-279
 - attenuation of GPG-279
 - falloff GPG-277, GPG-279
 - getting angle of GPG-279
 - soft GPG-279
 - spotlight effect GPG-126
 - spotlights
 - intensity of GPG-277
 - SPPlayer MPG-138
 - Spr GPG-88
 - Spr_ functions GPG-83
 - SPR_BOTTOMLEFT GPG-92
 - SPR_BOTTOMRIGHT GPG-92
 - SPR_CENTER GPG-92
 - spr_Colors attribute GPG-88
 - spr_Corners attribute GPG-86
 - Spr_Create function GPG-81, GPG-83
 - tags and values of GPG-89
 - Spr_CreateExtended function GPG-88
 - Spr_GetColors function GPG-91
 - Spr_GetCorners function GPG-91
 - Spr_GetDblendAttr function GPG-90
 - Spr_GetHeight function GPG-91
 - Spr_GetHSlice function GPG-91
 - Spr_GetPIP function GPG-90
 - Spr_GetPosition function GPG-91
 - Spr_GetTextureAttr function GPG-90
 - Spr_GetTextureData function GPG-90
 - Spr_GetVSlice function GPG-91
-

- Spr_GetWidth function GPG-90
- Spr_GetZValues function GPG-91
- spr_Height GPG-87
- spr_Height attribute GPG-86
- spr_HSlice GPG-87
- spr_HSlice attribute GPG-86
- Spr_MapCorners function GPG-92
- spr_Node GPG-87
- spr_Node attribute GPG-86
- spr_Position GPG-87
- spr_Position attribute GPG-86
- Spr_Remove functions GPG-91
- Spr_RemoveDbldAttr function GPG-91
- Spr_RemovePIP function GPG-91
- Spr_RemoveTextureAttr function GPG-91
- Spr_RemoveTextureData function GPG-91
- Spr_ResetCorners function GPG-92
- Spr_Rotate function GPG-81, GPG-92
- Spr_SetColors function GPG-90
- Spr_SetCorners function GPG-90
- Spr_SetDbldAttr function GPG-89
- Spr_SetGeometryEnable function GPG-93
- Spr_SetHeight function GPG-90
- Spr_SetHSlice function GPG-90
- Spr_SetPIP function GPG-90
- Spr_SetPosition function GPG-90
- Spr_SetTextureAttr function GPG-89
- Spr_SetTextureData function GPG-89
- Spr_SetVSlice function GPG-90
- Spr_SetWidth function GPG-90
- Spr_SetZValues function GPG-90
- spr_Skip attribute GPG-86
- spr_Skip flag GPG-87
- SPR_TAG_HEIGHT tag GPG-89
- SPR_TAG_HSLICE tag GPG-89
- SPR_TAG_NAME tag GPG-89
- SPR_TAG_PIP tag GPG-89
- SPR_TAG_PRIORITY tag GPG-89
- SPR_TAG_SKIP tag GPG-89
- SPR_TAG_TEXTUREDATA tag GPG-89
- SPR_TAG_VSLICE tag GPG-89
- SPR_TAG_WIDTH tag GPG-89
- SPR_TAG_XPOS tag GPG-89
- SPR_TAG_YPOS tag GPG-89
- spr_TexBlend GPG-87
- spr_TexBlend attribute GPG-86
- SPR_TOPLEFT GPG-92
- SPR_TOPRIGHT GPG-92
- Spr_Transform function GPG-92
- Spr_Translate function GPG-91
- spr_VSlice GPG-87
- spr_VSlice attribute GPG-86
- spr_Width GPG-87
- spr_Width attribute GPG-86
- spr_ZValues attribute GPG-88
- sprite GPG-83
 - 2D GPG-80
 - 2D, see also sprite object GPG-85
 - and Triangle Engine GPG-82
 - destination blend attribute of GPG-87
 - drawing GPG-83
 - node structure of GPG-87
 - Portfolio node GPG-87
 - TagArg pointer of GPG-88
 - texture blend attribute of GPG-87
- sprite node GPG-87
- sprite object GPG-82, GPG-85, GPG-86, GPG-88, GPG-92
 - and 2D Graphics Framework GPG-80
 - and cel object GPG-86
 - and TexBlend structure GPG-87
 - attributes of GPG-86
 - contents of GPG-89
 - corners of GPG-87
 - creating GPG-83, GPG-100
 - cutting into pieces GPG-87
 - defined GPG-80
 - displaying GPG-87
 - drawing GPG-83
 - extended, see extended sprite object GPG-80
 - extending GPG-88
 - geometry of GPG-93
 - height of GPG-87
 - manipulating position of GPG-91
 - parts of GPG-83
 - position of GPG-87
 - rendering GPG-93
 - rendering in strips and sections GPG-85
 - rotating GPG-83
 - scaling GPG-83
 - slicing GPG-86
 - tb_DAB element GPG-87
 - tb_PCB element GPG-87
 - tb_TAB element GPG-87
 - tb_TxData element GPG-87
 - width of GPG-87
- sprite object attribute functions GPG-89

- sprite object attribute settings
 - loading into Triangle Engine GPG-93
- sprite object attributes
 - removing GPG-91
 - retrieving GPG-90
 - setting GPG-89
- sprite object functions GPG-88
- sprite object geometry
 - disabling GPG-93
- sprite object positions GPG-91
- sprite objects, see also sprite GPG-82
- sprite, see also sprite object GPG-83
- sprite-based graphics GPG-1
- SpriteObj GPG-81, GPG-82, GPG-92
 - defined GPG-80
 - rendering in strips and sections GPG-85
- SpriteObj, see also sprite GPG-86
- spService() MPG-137, MPG-144
- spSetMarkerDecisionFunction() MPG-143
- SPSound MPG-138
- spt file, see .spt file DBG-22
- SQS2 compression VID-35
- square.dsp MPG-23
- SquashSnd
 - caveats TSD-27
 - compatible file types TSD-25
 - compression/decompression parameters TSD-26
 - convert raw file TSD-25
 - MPW tool TSD-25
 - parameters TSD-26
- squashsnd TSD-3
- SquashSound DSG-12, VID-11, TSD-75
- SrcBitmap attribute GPG-175
- SrcXOffset attribute GPG-175
- SrcYOffset attribute GPG-175
- SSB CDM-28, CDM-33, GPG-148, GPG-149, GPG-151, GPG-153, GPG-154, GPG-185, GPG-186, GPG-188, GPG-192, GPG-193
- SSND chunk DSG-17
- ssplCreateSoundSpooler() MPG-143
- ssplDeleteSoundSpooler() MPG-143
- ssplPlayData() MPG-147
- ssplProcessSignals() MPG-143
- ssplSpoolData() MPG-143, MPG-147
- ssplStartSpooler() MPG-143
- ssplStopSpooler() MPG-143
- stack
 - device PPG-105
- Stack Crawl menu item DBG-57, DBG-66
- Stack Crawl window DBG-47, DBG-57, DBG-66
- stack size PPG-25
- Stack window DBG-47, DBG-48
- standard dynamics processing
 - compression TSD-66
 - ducking TSD-66
 - expansion TSD-66
 - gating TSD-66
 - Spectral Dynamics command TSD-66
- Standard File dialog box DBG-52, DBG-62
- standard Macintosh functionality PRO-14
- StartAttachment() MPG-57
- starting a note MPG-173
- starting a stream DSG-34
- starting an releasing an instrument MPG-174
- StartInstrument() MPG-34
- StartMetronome() PPG-120
- StartMetronomeVBL() PPG-122
- StartObject() MPG-200, MPG-202
- StartReading TSD-44
- StartReadingAndPlaying TSD-44
- StartScoreNote() MPG-173
- StartSoundFile() MPG-131
- StartTime attribute GPG-282
- starving buffers DSG-54
- state information GPG-210
- static actions
 - in markers MPG-140
- Static attribute GPG-268
- Status attribute GPG-281
- Status menu item DBG-57, DBG-66
- Status window DBG-17, DBG-18, DBG-57, DBG-65, DBG-66
- step commands DBG-29
- Step In DBG-59, DBG-67
- Step In command DBG-29
 - using DBG-30
- Step In menu item DBG-59, DBG-67
- Step Over DBG-58, DBG-67
- Step Over command DBG-29
 - using DBG-31
- Step Over menu item DBG-58, DBG-67
- stepping through code DBG-29
- stereo vs. mono TSD-3
- Stereoscopic glasses PPG-102
- StickEventData PPG-198
- Stop menu item DBG-58, DBG-67
- Stop Process command TSD-70

- Stop Processing command TSD-70
- StopAttachment() MPG-58
- StopInstrument() MPG-38
- StopObject() MPG-200, MPG-202
- stopping a stream DSG-34
- Storage Manager Interface
 - displaying PPG-285
- Storage Manager interface PPG-281
- storage requestor object
 - deleting PPG-286
 - displaying PPG-285
 - modifying PPG-286
- StoreContextInfo() PPG-253, PPG-264, PPG-266
- STORREQ_CANCEL PPG-285
- STORREQ_OK PPG-285
- Stream DSG-6
- stream clock DSG-33
 - states DSG-34
- stream file
 - contents DSG-6
 - control chunk DSG-6
 - creating DSG-13
 - definition DSG-7
 - example DSG-6
 - header DSG-6
 - pausing DSG-34
 - playing DSG-16
 - preparing DSG-6
 - preparing (overview) DSG-6
 - starting DSG-34
 - stopping DSG-34
- stream header DSG-6
- stream header chunk DSG-42
- stream *See* stream file DSG-7
- stream time DSG-7
- streamblock
 - definition DSG-7
- streamblock size
 - optimizing size DSG-53
- streamblocksize DSG-53
- streamblocksize command DSG-15
- streambuffers DSG-15
- streamer thread DSG-20
 - initializing DSG-27
- streamerdeltapri DSG-15
- streaming TSD-7
- streaming audio data TSD-7
- streaming audio data with TSD-7
- string type
 - SDF GPG-232
- strips
 - rendering 2D sprites in GPG-85
- structure
 - EngLink GPG-283
- style argument
 - in 2D triangle primitives GPG-96
- Style attribute GPG-136, GPG-140
- subclass MPG-186
 - creating new class from GPG-289
 - defining GPG-289
 - surface GPG-293
 - user-defined GPG-290
- subclasses GPG-230, GPG-289
- subclassing
 - SDF classes that support GPG-295
- subclassing the Surface object GPG-293
- submixer
 - setting levels MPG-98
- subscriber DSG-15
 - connecting DSG-28
 - initializing DSG-28
- subscriber list DSG-17
- subscriber thread DSG-21
- subscriber threads
 - initializing DSG-28
- SubscriberChunkCommon DSG-43
 - common.chunkType field DSG-43
 - compressedVideo field DSG-45
 - framePeriod field DSG-44
 - maxPictureArea field DSG-44
 - version field DSG-44
- SubTimerTicks() PPG-123
- SubTimes() PPG-118
- Subtimes() PPG-122
- sub-Views GPG-214
- Sun sound file TSD-54
- SunsubscriberChunkCommon
 - compressedVideo field DSG-45
- superclass MPG-186
- Superheroine character GPG-262
- Supervisor Registers window DBG-46, DBG-57
- Surf_AddGeometry function GPG-143
- Surf_AddTriMesh function GPG-143
- Surf_Create function GPG-143
- Surf_Delete function GPG-143
- Surf_Display function GPG-143
 - implementing GPG-293

-
- Surf_FindGeometry function GPG-143, GPG-144
 - Surf_FuncCalcBound function GPG-293
 - Surf_FuncDisplay function GPG-293
 - Surf_GetBound function GPG-144
 - implementing GPG-293
 - SURF_None GPG-143
 - Surf_Touch function GPG-143
 - SURF_UseLast GPG-143
 - surface
 - adding new geometry to GPG-143
 - changing geometry of GPG-143
 - defining GPG-246
 - deleting GPG-117
 - displaying GPG-294
 - faceted GPG-246
 - hidden GPG-113
 - in SDF files GPG-245, GPG-248
 - previously constructed GPG-142
 - rendering GPG-142
 - replacing GPG-257
 - Surface attribute GPG-257
 - Surface class GPG-295
 - surface class GPG-234, GPG-293
 - surface color
 - defining GPG-246
 - Surface data structure
 - maintaining your own GPG-293
 - Surface geometry GPG-257
 - surface normal
 - defining GPG-246
 - Surface object GPG-256, GPG-107, GPG-141
 - architecture of GPG-142
 - subclassing GPG-293
 - surface objects
 - and Graphics Pipeline GPG-109
 - surface overrides GPG-293
 - surface subclass GPG-293
 - Surface type GPG-257, GPG-289
 - surfaces GPG-229, GPG-141
 - and SDF files GPG-231
 - creating GPG-143
 - destroying GPG-143
 - displaying GPG-143
 - previously compiled GPG-228
 - previously constructed GPG-106
 - rendering GPG-143
 - rendering of GPG-228, GPG-231, GPG-3
 - surfaces and textures GPG-150
 - sustain loop MPG-51, MPG-76
 - SwapBuffers function GPG-94, GPG-237
 - symbol file DBG-10
 - symbolic name GPG-230
 - symbolic names GPG-229
 - and SDF files GPG-229
 - in SDF files GPG-232, GPG-294
 - symbolic-information file DBG-10
 - symbols
 - DLL PPG-289
 - in modules PPG-289
 - Symbols menu item DBG-56, DBG-66
 - Symbols window DBG-56, DBG-66
 - SYNC CDM-4, CDM-9
 - synchronization TSD-6
 - display GPG-217
 - synchronization signal GPG-256
 - synchronizing audio and video TSD-5
 - syntax definitions TSD-94
 - synthesized audio TSD-76
 - synthesizer MPG-20
 - synthetic patch TSD-31
 - System 7 Pro PRO-2, PRO-3
 - System 7.1 DBG-viii, PRO-2, PRO-3
 - System 7.5 PRO-2, PRO-3
 - system events PPG-185
 - system layer DSG-41
 - system requirements DBG-vii, DBG-viii
 - system signals PPG-13
 - System Target Decoder DSG-41
 - system time
 - reading the current PPG-118
 - system-wide free memory pool PPG-58
- ## T
-
- ta_Tag PPG-32
 - table
 - color CDM-33, GPG-152
 - PIP, see also PIP table CDM-32, GPG-149
 - tag argument types PPG-70
 - tag arguments PPG-70
 - varArgs PPG-71
 - tag commands
 - TAG_JUMP PPG-32
 - TAG_NOP PPG-32
 - TAG_END PPG-32
 - TAG_ITEM_NAME GPG-224
 - TAG_ITEM_PRI PPG-21
 - TAG_JUMP PPG-32

- TAG_NOP PPG-32
- TagArg array GPG-213
- TagArg data structure PPG-70, PPG-71
- TagArg elements MPG-194
- TagArg pointer
 - of sprite object GPG-88
- TagArg values
 - applying MPG-195
- TagArgs
 - defined PPG-32
 - using PPG-31
- Tagged file format GSG-46
- Tags
 - special commands PPG-32
- tags PPG-31
- takeme folder CDM-12
- Target menu DBG-59, DBG-68
- target monitor program DBG-2
- Target Setup dialog box DBG-6
- Target | Memory Dump menu item DBG-42
- Target | Setup menu item DBG-7
- task
 - defined PPG-19
- task control block PPG-2, PPG-3
- Task menu item DBG-57, DBG-66
- task priorities PPG-3
 - execution of PPG-3
- task states PPG-3
 - ready to run PPG-3
 - running PPG-3
 - waiting PPG-3
- Task Symbols window
 - printing DBG-63
- Task window DBG-57, DBG-66
 - using DBG-43
- tasks
 - changing ownership PPG-23
 - changing priorities PPG-23
 - child PPG-5
 - closing PPG-4
 - communication among PPG-26
 - controlling state PPG-22
 - creating PPG-20
 - ending PPG-21
 - intertask communications PPG-13
 - parent PPG-4
 - spawning PPG-5
 - starting on bootup PPG-20
 - thread PPG-5
 - yielding PPG-22
- tb_DAB attribute GPG-86
- tb_DAB element
 - and sprites GPG-87
- tb_PCB attribute GPG-86
- tb_PCB element
 - and sprite objects GPG-87
- tb_TAB attribute GPG-86
- tb_TAB element
 - and sprites GPG-87
- tb_TxData attribute GPG-86
- tb_TxData element
 - and sprite objects GPG-87
- TCB PPG-2, PPG-3, PPG-14, PPG-85
- TCB data structure PPG-20
- TCM_BORDERED flag GPG-97
- TCM_FULL flag GPG-98
- TE, see also Triangle Engine CDM-61, GPG-3, GPG-148, GPG-175
- tear-free rendering CDM-13
- TEContext GPG-210
 - and multitasking GPG-211
 - configuration options of GPG-211
 - contents of GPG-210
- TEContext Item
 - described GPG-210
- TEContext item GPG-197, GPG-211
 - allocating memory for GPG-211
 - creating GPG-211
 - deleting GPG-211
 - memory for GPG-211
- TEContext Item structure GPG-211
- Telecine VID-14, VID-2
- Telecine process DSG-10, VID-15
 - changing frame rate VID-38
 - inserting composite frames VID-38
 - understanding VID-15
- template
 - deleting MPG-39
- tempo
 - setting MPG-168
- Terminal menu item DBG-57, DBG-66
- Terminal Portfolio window
 - starting a debugging session DBG-16
- Terminal window
 - described DBG-9
- Terminal window, see Portfolio Terminal window DBG-9

- TermJuggler() MPG-180, MPG-206
- TermScoreMixer() MPG-179
- testing TSD-6
- Testing performance CDM-43
- Tex GPG-164
- Tex_Copy function GPG-151, GPG-163
- Tex_Create function GPG-163
- Tex_Delete function GPG-151, GPG-163
- Tex_GetDepth GPG-164
- Tex_GetFormat function GPG-164
- Tex_GetHeight function GPG-164
- Tex_GetMaxHeight function GPG-164
- Tex_GetMaxWidth function GPG-164
- Tex_GetMinHeight function GPG-164
- Tex_GetMinWidth function GPG-164
- Tex_GetNumLOD function GPG-151, GPG-163
- Tex_GetTexelData GPG-163
- Tex_GetTexelData function GPG-164
- Tex_GetWidth function GPG-164
- Tex_Load function GPG-151, GPG-163
- Tex_SetDepth function GPG-164
- Tex_SetFormat function GPG-164
- Tex_SetMinHeight function GPG-164
- Tex_SetMinWidth function GPG-164
- Tex_SetNumLOD function GPG-151
- Tex_SetTexelData function GPG-163
- texarray class GPG-247
- TexBlend array GPG-247
- TexBlend arrays GPG-109
- TexBlend attribute GPG-112, GPG-114
- TexBlend attributes GPG-152, GPG-160
 - getting and setting GPG-172
- TexBlend class
 - in SDF files GPG-244
- TexBlend coordinates
 - clamping GPG-161
- TexBlend functions GPG-154
- TexBlend object GPG-257, GPG-258, GPG-107, GPG-144, GPG-159, GPG-161
 - architecture of GPG-159
 - creating GPG-159
 - defined GPG-152
 - described GPG-159
 - in runtime texture carving GPG-167
 - loading GPG-159
- TexBlend objects GPG-229, GPG-109
 - creating and managing GPG-165
 - in SDF files GPG-242
- TexBlend offsets
 - setting and getting GPG-169
- TexBlend structure
 - and sprite objects GPG-87
- TexCoord attribute GPG-136
- texcoord type
 - SDF GPG-232
- TexCoordMode GPG-97, GPG-98
- texcow.csf DBG-26
- texcow.csf file DBG-15
- texel CDM-18, GPG-148
 - alpha component CDM-28, GPG-148
 - blending CDM-34, GPG-158
 - color component CDM-28, GPG-148
 - color values of GPG-149
 - components of CDM-28, CDM-34, GPG-148, GPG-151, GPG-154
 - compressed GPG-148
 - compression of CDM-28, GPG-148
 - defined CDM-18, GPG-148
 - expansion format of GPG-164
 - interpreting color data in CDM-33, GPG-152
 - mapping to pixels CDM-19, GPG-145
 - uncompressed GPG-148
- texel color values
 - kinds of GPG-149
- texel data
 - getting GPG-163
 - setting GPG-163
- texel data area GPG-164
- TexLOD
 - definition of GPG-163
- texOffset attribute GPG-243
- texScale attribute GPG-243
- text
 - deleting DBG-55, DBG-64
 - editing DBG-54, DBG-64
 - selecting DBG-55, DBG-64
- texture GPG-256
 - address of CDM-18
 - and TEContext Item GPG-210
 - coordinate system of CDM-18
 - copying GPG-163
 - creating GPG-163
 - defined CDM-17, GPG-144
 - deleting GPG-163
 - getting depth of GPG-164
 - loading GPG-163
 - matching with material GPG-256
 - maximum size of GPG-148

- setting depth of GPG-164
- size of GPG-150
- storing in memory GPG-148, GPG-151
- uncompressed GPG-149
- Texture API GPG-109, GPG-144
- texture application blending CDM-36, GPG-158, GPG-159
- texture array GPG-151
 - obtaining GPG-258
- texture attribute settings
 - 2D GPG-95
- texture blend attribute
 - and sprites GPG-87
- texture carving
 - runtime, see also runtime texture carving GPG-166
- Texture Coordinates attribute GPG-140
- texture coordinates GPG-114
- texture filtering CDM-35, GPG-159
 - example of CDM-24, GPG-157
 - mipmap CDM-19, GPG-146
- texture format GPG-148
- texture functions GPG-144, GPG-163, GPG-173
- texture generation GPG-242
- texture generation definition GPG-243
- texture load rectangle, see also load rectangle GPG-159, GPG-160
- texture mapping CDM-19, GPG-106, GPG-145
- texture mapping step by step CDM-35, GPG-159
- Texture object GPG-141, GPG-150
 - components of GPG-150
 - constructing GPG-151
 - format of GPG-151
 - initializing GPG-150
- texture objects
 - and Graphics Pipeline GPG-109
- texture of sprite object
 - sprite object
 - texture of GPG-87
- texture opacity GPG-151
- texture RAM GPG-211
- texture RAM, see also TRAM CDM-28, GPG-211, GPG-148
- texture-mapping
 - example of CDM-33, GPG-153
- texture-rendering hardware GPG-147
- Textures
 - tiled CDM-6
- textures GPG-232, GPG-3, GPG-109, GPG-144, GPG-173
 - and SDF files GPG-229
 - binding to surface GPG-293
 - changing GPG-258
 - creating and managing GPG-144
 - in TexBlend arrays GPG-247
 - shuffling GPG-258
 - storing CDM-18
- textures and surfaces GPG-150
- Textures attribute GPG-257
- textures in SDF files GPG-242
- texturing GPG-117
- The GPG-258
- this DBG-24
- thread
 - creating PPG-73
- Threads PPG-20
- threads PPG-5, PPG-24
 - ending PPG-26
 - launching DSG-26
 - priorities DSG-54
 - returning memory on death PPG-26
 - starting PPG-24
- threadSig1 PPG-84
- threadSig2 PPG-84
- three-dimensional, see 3D GPG-1
- Threshold Level option TSD-68
- ticks MPG-189
- time
 - reading PPG-118
 - waiting for PPG-118, PPG-119
- Time Scale option TSD-65
- TimeLaterThan() PPG-123
- TimeLaterThanOrEqual() PPG-123
- timer device PPG-115
 - waiting for specific time PPG-118
- timer devices PPG-117
- timer IOREq
 - creating PPG-117
- TIMER_CMD_GETTIME_VBL PPG-121
- TIMER_UNIT_VBLANK PPG-117
- TIMERCMD_DELAY PPG-119
- TIMERCMD_DELAY_VBL PPG-121
- TIMERCMD_DELAYUNTIL_USEC PPG-118
- TIMERCMD_DELAYUNTIL_VBL PPG-121
- TIMERCMD_METRONOME PPG-120, PPG-122
- TimerTicksLaterThan() PPG-123
- TimerTicksLaterThanOrEqual() PPG-123
- TimeVal structure PPG-117

- timing MPG-40
 - audio ticks MPG-40
 - metronome PPG-120
- timing hardware PPG-117
- timing signals GPG-256
- tools
 - conversion GPG-256
- Tools menu DBG-60, DBG-69
- TopEdge GPG-210
- torus
 - drawing GPG-294
- Trace
 - AddTrace()
 - traceWrap DSG-57
 - DumpRawTraceBuffer() DSG-57
 - DumpTraceCompletionStats(DSG-57
- trackball PPG-102
- trackball return data PPG-196
- TRACKED_SIZE PPG-54
- TRAM CDM-33, GPG-211, GPG-150, GPG-153
 - and TEContext item GPG-211
 - and Triangle Engine GPG-148
- TRAM (texture RAM) CDM-28, GPG-148
- Trans GPG-130
- Trans_Add function GPG-130, GPG-131
- Trans_AxisRotate function GPG-130, GPG-132
- Trans_Copy function GPG-130, GPG-131
- Trans_Create function GPG-109, GPG-129, GPG-130
- Trans_Delete function GPG-129
- Trans_Frustum function GPG-135
- Trans_GetData function GPG-129, GPG-130
- Trans_GetInverse function GPG-131
- Trans_Identity function GPG-129, GPG-130
- Trans_Init function GPG-129
- Trans_Inverse function GPG-131
- Trans_Invert function GPG-130
- Trans_LookAt function GPG-130, GPG-133
- Trans_Mul function GPG-130, GPG-131
- Trans_Perspective function GPG-130, GPG-134
- Trans_Polarview function GPG-130
- Trans_PolarviewLookAt function GPG-133
- Trans_Pop function GPG-135
- Trans_PostMul function GPG-130, GPG-131
- Trans_PreMul function GPG-130
 - pre-multiplying GPG-131
- Trans_Print function GPG-135
- Trans_Rotate function GPG-130, GPG-131
- Trans_Scale function GPG-130, GPG-132
- Trans_SetData function GPG-129, GPG-130, GPG-131
- Trans_Subtract function GPG-130, GPG-131
- Trans_Touch function GPG-130
- Trans_Translate function GPG-249, GPG-130, GPG-132
- Trans_Transpose function GPG-130, GPG-131
- TRANS_XAxis GPG-246, GPG-247, GPG-131
- Trans_xxx functions GPG-131
- TRANS_YAxis GPG-246, GPG-247, GPG-131
- TRANS_ZAxis GPG-246, GPG-247, GPG-131
- transferring memory PPG-58
- Transform GPG-239
- transform
 - allocating GPG-129
 - character GPG-292
 - initializing GPG-129
 - inverse GPG-131
- Transform attribute GPG-238, GPG-239
- Transform matrix GPG-135
- Transform object GPG-252, GPG-129, GPG-135
- transformation GPG-242, GPG-249, GPG-263, GPG-292, GPG-2, GPG-3, GPG-129, GPG-134
 - character GPG-238, GPG-242
 - global GPG-242
 - local GPG-265
 - matrix GPG-239, GPG-292
 - rotate GPG-132
 - scale GPG-133
 - translate GPG-132
 - world GPG-265
- transformation equations GPG-120
- transformation matrix GPG-249, GPG-252, GPG-254, GPG-119
 - changing GPG-130
 - deleting GPG-129
 - for a character GPG-248
- transformation operation functions GPG-129
- transformation operations GPG-249
 - character GPG-238
- transformations GPG-129, GPG-141
 - geometric GPG-129
 - geometry pipeline GPG-135
 - order-sensitive GPG-249
- transformations on points GPG-119
- transformations on vectors GPG-119, GPG-120
- transforming GPG-1, GPG-3, GPG-106
- transforming characters GPG-247
- transforming characters with matrices GPG-249

- transforms
 - animation and GPG-280
 - creating GPG-109
- Translate GPG-247
- translate transformation GPG-132
- translation GPG-249
- transparency
 - of 3D objects CDM-17, GPG-144
 - of scene GPG-271
 - setting GPG-271
- Transparent attribute GPG-268
- traversing a linked list PPG-43
- traversing a list
 - from back to front PPG-44
 - from front to back PPG-43
- Triangle Disassembly menu item DBG-57, DBG-66
- Triangle Engine GPG-1, GPG-3
 - and 2D attributes GPG-95
 - and 2D Graphics Framework GPG-79
 - and 2D rendering GPG-96
 - and destination blending GPG-173
 - and Graphics Pipeline GPG-105
 - and GState object GPG-82
 - and multitasking GPG-210
 - and PIP tables GPG-151
 - and sprite object geometry GPG-93
 - and sprite objects GPG-82
 - and TEContext Item GPG-210
 - and TRAM GPG-148
 - color calculations performed by CDM-61, GPG-175
 - deferred mode CDM-4
 - defined GPG-1
 - described GPG-256, GPG-3
 - Destination Blender CDM-2
 - Graphics Pipeline and GPG-2
 - state registers
 - shifts and masks CDM-7
 - Texture Mapper CDM-2
- Triangle Engine Disassembly window DBG-48, DBG-57, DBG-66
- Triangle Engine hardware context, see also TEContext GPG-210
- Triangle Engine PIP GPG-154
- triangle fan
 - drawing GPG-115
- triangle fan primitive
 - 2D GPG-96
- triangle list
 - rendering GPG-115
- triangle primitives
 - 2D GPG-95, GPG-96
- triangle strip
 - drawing GPG-116
- triangle strip primitive
 - 2D GPG-96
- triangles GPG-3, GPG-113
 - 2D GPG-95
 - culling GPG-113
 - disjoint GPG-95
- TriFan GPG-143
- trigger masks PPG-176, PPG-184, PPG-188, PPG-190
- trigger points MPG-51
 - release MPG-51
 - start MPG-51
 - stop MPG-51
- tri-linear filtering GPG-192
 - quasi, see also quasi tri-linear filtering CDM-26, GPG-147, GPG-156
- TriList GPG-143
- TriStrip GPG-143
- TuneInsTemplate() MPG-92
- TuneInstrument() MPG-92
- tuning MPG-90, MPG-91
 - applying to instruments MPG-92
 - creating MPG-91
 - deleting MPG-92
 - extending MPG-91
 - octaves MPG-91
- Turn function GPG-246
- turn functions GPG-244
 - character GPG-244
- TweakKnob() MPG-71
- two-dimensional, see also 2D GPG-1
- two-sided lighting GPG-128
- TX_AlphaSelectConstAlpha constant GPG-172
- TX_AlphaSelectPrimAlpha constant GPG-172
- TX_AlphaSelectTexAlpha constant GPG-172
- TX_Bilinear constant GPG-171
- TX_BlendOpLerp constant GPG-171
- TX_BlendOpMult constant GPG-171
- TX_BlendOutSelectBlend constant GPG-172
- TX_BlendOutSelectPrim constant GPG-172
- TX_BlendOutSelectTex constant GPG-172
- TX_ColorSelectConstAlpha constant GPG-172
- TX_ColorSelectConstColor constant GPG-172

-
- TX_ColorSelectPrimAlpha constant GPG-172
 - TX_ColorSelectPrimColor constant GPG-172
 - TX_ColorSelectTexAlpha constant GPG-172
 - TX_ColorSelectTexColor constant GPG-172
 - TX_Linear constant GPG-171
 - TX_Nearest constant GPG-171
 - TX_PipSelectColorTable constant GPG-170
 - TX_PipSelectConst constant GPG-170
 - TX_PipSelectTexture constant GPG-170
 - TX_QuasiTrilinear constant GPG-171
 - TX_WrapModeClamp GPG-168
 - TX_WrapModeTile GPG-168
 - Txb_Copy function GPG-166
 - Txb_Create function GPG-159, GPG-165
 - Txb_Delete function GPG-165
 - Txb_GetBlendColorSSB0 function GPG-154
 - Txb_GetDbIAInputSelect function GPG-183
 - Txb_GetDbIAAlpha0ClampControl function GPG-191
 - Txb_GetDbIAAlpha1ClampControl function GPG-191
 - Txb_GetDbIAAlphaFracClampControl function GPG-192
 - Txb_GetDbIALUOperation function GPG-190
 - Txb_GetDbIAMultCoefSelect function GPG-184
 - Txb_GetDbIAMultConstControl function GPG-185
 - Txb_GetDbIAMultConstSSB0 function GPG-185
 - Txb_GetDbIAMultConstSSB1 function GPG-186
 - Txb_GetDbIAMultRtJustify function GPG-186
 - Txb_GetDbIBInputSelect function GPG-187
 - Txb_GetDbIBMultCoefSelect function GPG-188
 - Txb_GetDbIBMultConstControl function GPG-188
 - Txb_GetDbIBMultConstSSB0 function GPG-188
 - Txb_GetDbIBMultConstSSB1 function GPG-188
 - Txb_GetDbIBMultRtJustify function GPG-190
 - Txb_GetDbIDestAlphaConstSelect function GPG-192
 - Txb_GetDbIDestAlphaConstSSB0 function GPG-192
 - Txb_GetDbIDestAlphaConstSSB1 function GPG-193
 - Txb_GetDbIDestAlphaSelect function GPG-192
 - Txb_GetDbIDiscard function GPG-182
 - Txb_GetDbIDitherMatrixA function GPG-178
 - Txb_GetDbIDitherMatrixB function GPG-178
 - Txb_GetDbIDSBConst function GPG-182
 - Txb_GetDbIDSBSelect function GPG-182
 - Txb_GetDbIEnableAttrs function GPG-181
 - Txb_GetDbIFinalDivide function GPG-190
 - Txb_GetDbIRGBConstIn function GPG-183
 - Txb_GetDbISrcBaseAddr function GPG-193
 - Txb_GetDbISrcPixels32Bit function GPG-193
 - Txb_GetDbISrcXStride function GPG-193
 - Txb_GetDbISrcYOffset function GPG-194
 - Txb_GetDbIXWinClipMax function GPG-179
 - Txb_GetDbIXWinClipMin function GPG-178
 - Txb_GetDbIYWinClipMax function GPG-180
 - Txb_GetDbIYWinClipMin function GPG-180
 - Txb_GetDbIZCompareControl function GPG-176
 - Txb_GetDbIZXOffset function GPG-176
 - Txb_GetDbIZYOffset function GPG-177
 - Txb_GetFirstLOD function GPG-167
 - Txb_GetInterFilter GPG-161
 - Txb_GetLoadRect function GPG-167
 - Txb_GetMagFilter function GPG-160
 - Txb_GetMinFilter function GPG-160
 - Txb_GetNumLOD function GPG-162, GPG-168
 - Txb_GetPip function GPG-166
 - Txb_GetTexture function GPG-166
 - Txb_GetTramOffset function GPG-169
 - Txb_GetTxAlphaOut function GPG-161, GPG-173
 - Txb_GetTxBlendColorSSB0 function GPG-173
 - Txb_GetTxBlendColorSSB1 function GPG-173
 - Txb_GetTxBlendOp function GPG-171
 - Txb_GetTxColorOut function GPG-161, GPG-173
 - Txb_GetTxFirstAlpha function GPG-161, GPG-172
 - Txb_GetTxFirstColor function GPG-171
 - Txb_GetTxInterFilter function GPG-172
 - Txb_GetTxMagFilter function GPG-172
 - Txb_GetTxMinFilter function GPG-172
 - Txb_GetTxPipAlphaSel function GPG-170
 - Txb_GetTxPipColorSel function GPG-170
 - Txb_GetTxPipConstSSB0 function GPG-171
 - Txb_GetTxPipConstSSB1 function GPG-171
 - Txb_GetTxPipIndexOffset function GPG-171
 - Txb_GetTxPipSSBSel function GPG-171
 - Txb_GetTxSecondAlpha function GPG-161, GPG-172
 - Txb_GetTxSecondColor function GPG-171
 - Txb_GetTxThirdColor function GPG-171
 - Txb_GetXOffset function GPG-169
 - Txb_GetXSize function GPG-170
 - Txb_GetXWrap function GPG-161, GPG-168
 - Txb_GetYOffset function GPG-169
 - Txb_GetYSize function GPG-170
 - Txb_Load function GPG-159, GPG-166
 - Txb_SetBlendColorSSB0 function GPG-154
 - Txb_SetBlendColorSSB1 function GPG-154
-

Txb_SetDblAlphaInputSelect function GPG-183
Txb_SetDblAlpha0ClampControl function GPG-191
Txb_SetDblAlpha1ClampControl function GPG-191
Txb_SetDblAlphaFracClampControl function GPG-192
Txb_SetDblALUOperation function GPG-190
Txb_SetDblAMultCoefSelect function GPG-184
Txb_SetDblAMultConstControl function GPG-185
Txb_SetDblAMultConstSSB0 function GPG-185
Txb_SetDblAMultConstSSB1 function GPG-186
Txb_SetDblAMultRtJustify function GPG-186
Txb_SetDblBInputSelect function GPG-187
Txb_SetDblBMultCoefSelect function GPG-188
Txb_SetDblBMultConstControl function GPG-188
Txb_SetDblBMultConstSSB0 function GPG-188
Txb_SetDblBMultConstSSB1 function GPG-188
Txb_SetDblBMultRtJustify function GPG-190
Txb_SetDblDestAlphaConstSelect function GPG-192
Txb_SetDblDestAlphaConstSSB0 function GPG-192
Txb_SetDblDestAlphaConstSSB1 function GPG-193
Txb_SetDblDestAlphaSelect function GPG-192
Txb_SetDblDiscard function GPG-182
Txb_SetDblDitherMatrixA function GPG-178
Txb_SetDblDitherMatrixB function GPG-178
Txb_SetDblDSBConst function GPG-182
Txb_SetDblDSBSelect function GPG-182
Txb_SetDblEnableAttrs function GPG-181
Txb_SetDblFinalDivide function GPG-190
Txb_SetDblRGBConstIn function GPG-183
Txb_SetDblSrcBaseAddr function GPG-193
Txb_SetDblSrcXStride function GPG-193
Txb_SetDblSrcYOffset function GPG-194
Txb_SetDblXWinClipMax function GPG-179
Txb_SetDblXWinClipMin function GPG-178
Txb_SetDblYWinClipMax function GPG-180
Txb_SetDblYWinClipMin function GPG-180
Txb_SetDblZCompareControl function GPG-176
Txb_SetDblZXOffset function GPG-176
Txb_SetDblZYOffset function GPG-177
Txb_SetFirstLOD function GPG-162, GPG-167
Txb_SetInterFilter function GPG-161
Txb_SetLoadRect function GPG-160, GPG-162, GPG-167
Txb_SetMagFilter function GPG-160
Txb_SetMinFilter function GPG-160
Txb_SetNumLOD function GPG-162, GPG-167
Txb_SetPip function GPG-166
Txb_SetTexture function GPG-166
Txb_SetTramOffset function GPG-169
Txb_SetTxAlphaOut function GPG-161, GPG-173
Txb_SetTxBlendColorSSB0 function GPG-173
Txb_SetTxBlendColorSSB1 function GPG-173
Txb_SetTxBlendOp function GPG-171
Txb_SetTxColorOut function GPG-161, GPG-173
Txb_SetTxFirstAlpha function GPG-161, GPG-172
Txb_SetTxFirstColor function GPG-171
Txb_SetTxInterFilter function GPG-172
Txb_SetTxMagFilter function GPG-172
Txb_SetTxMinFilter function GPG-172
Txb_SetTxPipAlphaSel function GPG-170
Txb_SetTxPipColorSel function GPG-170
Txb_SetTxPipConstSSB0 function GPG-154, GPG-171
Txb_SetTxPipConstSSB1 function GPG-154, GPG-171
Txb_SetTxPipIndexOffset function GPG-171
Txb_SetTxPipSSBSel function GPG-154, GPG-171
Txb_SetTxSecondAlpha function GPG-161, GPG-172
Txb_SetTxSecondColor function GPG-171
Txb_SetTxThirdColor function GPG-171
Txb_SetXOffset function GPG-169
Txb_SetXSize function GPG-169
Txb_SetXWrap function GPG-161, GPG-168
Txb_SetYOffset function GPG-169
Txb_SetYSize function GPG-170
Txb_SetYWrap function GPG-161, GPG-168
TxInterFilter attribute GPG-161
TxMagFilter attributes GPG-160
TxMinFilter attribute GPG-160
Txt_GetLoadRect GPG-160
TXTR FORM PPG-245, PPG-260
type
 of an object GPG-111
Type attribute GPG-136, GPG-140
types
 character GPG-293
 SDF GPG-234
typographical conventions DBG-viii

U

u coordinate CDM-18
 UI events PPG-185
 uint16 TSD-90
 uint32 CDM-34, GPG-154, TSD-90
 uint7 TSD-90
 uint8 TSD-90
 uncompressed texture GPG-149
 Unicode PPG-218
 uniform texture format, see UTF GPG-163
 UnimportByAddress() PPG-298
 UnimportByName() PPG-298
 UniversalInsertNode() PPG-39, PPG-42
 UNIX PPG-140
 UnloadIcon() PPG-272, PPG-274
 UnloadInstrument() MPG-39
 UnloadPIMap() MPG-179
 UnloadSample() MPG-54
 UnloadSoundFile() MPG-134
 UnlockDisplay function GPG-224
 UnlockSemaphore() PPG-12, PPG-78, PPG-80
 unnamed objects GPG-232
 Unwrapped file GSG-46
 updating GPG-224
 user context
 specifying MPG-166
 user interface DBG-1
 user registers DBG-44
 User Registers window DBG-43, DBG-44, DBG-57, DBG-66
 user-defined subclass GPG-290
 using TSD-40
 using binaural filter TSD-58
 using MIDI functions MPG-172
 UTF GPG-247
 UTF (unified texel format) file DBG-14
 UTF (unified texture format) file GPG-151
 UTF (uniform texture format) file GPG-163
 UTF files GPG-242

V

v coordinate CDM-18
 v_Bitmap field GPG-206
 v_BitmapItem field GPG-206
 v_Height GPG-207
 v_Height field GPG-205
 v_LeftEdge GPG-223

v_LeftEdge field GPG-205
 v_PixHeight GPG-207
 v_PixHeight field GPG-206
 v_PixWidth GPG-207
 v_PixWidth field GPG-206
 v_TopEdge GPG-223
 v_TopEdge field GPG-205
 v_Type field GPG-206
 v_Width GPG-207
 v_Width field GPG-205
 v_WinLeft field GPG-206
 v_WinTop field GPG-206
 ValidateDate() PPG-174
 ValidateObject() MPG-188
 Value field DBG-37
 Value text box DBG-37
 varargs PPG-34
 VarArgs Tag Arguments PPG-71
 variable
 changing the value of DBG-34, DBG-36
 determining the value of DBG-34
 evaluating DBG-34
 examining the value of DBG-34, DBG-35
 variables
 working with DBG-32
 Variables menu item DBG-56, DBG-66
 Variables window DBG-32, DBG-36, DBG-37, DBG-56, DBG-58, DBG-66
 adding to display DBG-33
 and arrays DBG-38
 changing values interactively DBG-33
 deleting a variable DBG-33
 dereferencing a pointer DBG-34
 features of DBG-33
 selecting for receiving data DBG-34
 two ways to use DBG-34
 working with DBG-33
 Variables windows
 multiple DBG-33
 Varispeed box TSD-69
 Varispeed command TSD-68
 Varispeed dialog TSD-69
 Quality TSD-69
 Varispeed Function TSD-69
 Varspeed box TSD-69
 Vary by Pitch/Scale TSD-69
 Varispeed Edit Function dialog TSD-68, TSD-69
 Varispeed Function option TSD-69
 vblank PPG-120

- vblank count
 - reading PPG-121
 - waiting for PPG-121
- vblank interval
 - waiting for PPG-121
- VBlankTimeVal structure PPG-120
- VDIG VID-16
- VDL GPG-219, GPG-220, GPG-221, GPG-224, GPG-225
- VDL (video display list) GPG-223
 - defined GPG-195
- VDL hardware
 - and Views GPG-204
- VDL instructions GPG-223
- Vec3_Add function GPG-119, GPG-121
- Vec3_Cross function GPG-119, GPG-122
- Vec3_Dot GPG-122
- Vec3_Dot function GPG-119
- Vec3_Equal function GPG-121
- Vec3_Length function GPG-119, GPG-122
- Vec3_LengthSquared function GPG-119, GPG-122
- Vec3_Negate function GPG-119, GPG-121
- Vec3_Normalize function GPG-119, GPG-122
- Vec3_Print function GPG-119, GPG-122
- Vec3_Set function GPG-119, GPG-121
- Vec3_Sub function GPG-121
- Vec3_Subtract function GPG-119
- Vec3_Transform function GPG-119, GPG-122
- vector
 - column GPG-120
 - getting length of GPG-122
 - getting squared length of GPG-122
 - initializing GPG-121
 - negating GPG-121
 - normalizing GPG-122
 - row GPG-119
 - subtracting GPG-121
- vector functions GPG-121
- Vector Quantization VID-23
- Vector3 structure GPG-118
- vectors GPG-118
 - adding GPG-121
 - cross-product of GPG-122
 - multiplying GPG-122
 - operations on GPG-121
 - transformations on GPG-119, GPG-120
 - transforming GPG-122
- versionstamp TSD-91
- vertex GPG-124
 - color of GPG-124
 - components of GPG-138
- vertex color GPG-114
- vertex data
 - arrays of GPG-114
 - deleting GPG-138
 - representing GPG-136
- vertex data areas
 - allocating GPG-138
- vertex header instruction CDM-5
- vertex locations
 - transforming GPG-139
- vertex normals GPG-114
- vertex style
 - obtaining name of GPG-138
- VertexOrder GPG-241
- vertical blank GPG-259
- vertical blank timer PPG-120
- vertices
 - clearing GPG-139
 - homogeneous GPG-120
- verticesPerColumn field GPG-241
- verticesPerRow field GPG-241
- video
 - compressing DSG-12
 - converting DSG-12
 - generating DSG-10
- video acquisition card VID-16
- video capture
 - requirements VID-15
 - video acquisition card VID-16
 - video controller card VID-16
- video capture process
 - compression during capture VID-17
 - requirements VID-16
- video controller card VID-16
- video digitizing VID-13
- video display list, see VDL GPG-195, GPG-219
- Video Explorer card VID-16
- video formats
 - Betacam VID-4
 - D1 VID-4
- video logic GPG-260
- video processing paths
 - choosing VID-3
 - factors VID-3
- video processing tools VID-37
- video standards GPG-257

-
- video tape VID-2
 - View
 - abstract GPG-204
 - attaching to a display GPG-216
 - changing dimensions of GPG-223
 - characteristics of GPG-204
 - creating (example code) GPG-208
 - default settings of GPG-207
 - described GPG-204
 - enabling signaling for GPG-217
 - Graphics Folio GPG-209
 - modifying GPG-224
 - moving GPG-223
 - removing from a display GPG-217
 - removing from parent ViewList GPG-216
 - view
 - direction of GPG-133
 - field of GPG-134
 - View coordinate system GPG-204
 - View hierarchy GPG-215
 - View item GPG-197, GPG-205, GPG-207
 - creating GPG-205
 - modifying GPG-205
 - View item fields GPG-205
 - View item structure GPG-205
 - view matrix stack GPG-135
 - View menu DBG-55, DBG-65
 - View properties GPG-206
 - View signal
 - described GPG-217
 - View signals
 - types of GPG-217
 - using GPG-217, GPG-222
 - view signals GPG-217
 - View type GPG-204
 - view volume
 - getting height of GPG-274
 - perspective GPG-135
 - setting height of GPG-274
 - View | Breakpoints menu command DBG-29
 - View | FP Registers menu command DBG-45
 - View | PPC Users menu command DBG-44
 - View | Send Variable Data menu item DBG-36
 - View | Stack Crawl menu command DBG-47
 - View | Status menu item DBG-17
 - View | Supervisor Registers menu command DBG-47
 - View | Task menu command DBG-43, DBG-44, DBG-45, DBG-47
 - View | Terminal menu item DBG-9
 - View | Triangle Disassembly menu command DBG-49
 - View | Variables menu command DBG-39
 - View | Variables menu item DBG-32
 - viewing angle GPG-272
 - viewing position GPG-133
 - viewing pyramid GPG-134
 - viewing window GPG-271
 - ViewList GPG-214, GPG-225
 - attaching Views to GPG-224
 - children of GPG-215
 - creating GPG-210
 - defined GPG-210
 - example code GPG-216
 - modifying GPG-210, GPG-224
 - removing a View from GPG-216
 - tags GPG-210
 - viewlist GPG-82
 - ViewList hierarchy GPG-215
 - ViewList item GPG-197
 - ViewList Item structure GPG-210
 - ViewLists GPG-210, GPG-224
 - and Views GPG-210
 - viewpoint GPG-133, GPG-134
 - viewport GPG-113, GPG-134
 - Views GPG-204, GPG-209, GPG-214
 - and ViewLists GPG-210
 - assigning names of GPG-225
 - creating GPG-215
 - managing GPG-224
 - optimizing compilation of GPG-224
 - overlapping GPG-214
 - reordering GPG-216
 - using with ViewLists GPG-224
 - visible GPG-215
 - views and viewlists
 - hierarchy of GPG-214
 - VIEWTAG_BESILENT GPG-223
 - VIEWTAG_BITMAP GPG-225
 - ViewVol attribute GPG-273
 - virtual MIDI device MPG-153
 - visibility
 - of a scene GPG-270
 - setting GPG-270
 - Visible attribute GPG-238, GPG-239, GPG-268
 - visible character attribute GPG-238
 - voice MPG-154
 - defining maximum MPG-159
-

- voice allocation
 - dynamic MPG-214
 - freeing instruments created by MPG-171
- voices MPG-15
 - maximum number of MPG-160
- volume
 - bounding, see also bounding volume GPG-252
 - bounding, see also bounding volume GPG-239
 - view GPG-135
- volume settings TSD-7
- VQ VID-23

W

- wait queue PPG-4
- wait state PPG-85, PPG-22
- waiting
 - task state PPG-3
- waiting for signals PPG-85
- waiting queue PPG-3
- waiting tasks PPG-4
- WaitIO() PPG-110, PPG-4
- WaitPort() PPG-111, PPG-4, PPG-15, PPG-92
- WaitSignal() PPG-111, PPG-4, PPG-14, PPG-85, PPG-86, PPG-22, PPG-23, PPG-26
- WaitTime() PPG-119
- WaitTimeVBL() PPG-121
- WaitUntil () PPG-118
- WaitUntil() PPG-118
- WaitUntilVBL() PPG-121
- WaitVBL function GPG-219
- weave into stream TSD-31
- weave script VID-53
- Weaver DSG-13, VID-7, VID-8, VID-11, VID-12, VID-28, VID-49
 - output DSG-16
 - using DSG-16
- Weaver script DSG-6
 - additions for audio DSG-29
 - example DSG-14
 - starting and stopping streams DSG-35
- weaving VID-36
- whatToDo field DSG-26
- width
 - of sprite object GPG-87
- WinClipInEnable CDM-63, GPG-178
- WinClipInEnanble GPG-178
- WinClipOutEnable CDM-63, GPG-178
- window
 - clip GPG-178
 - clipping GPG-179, GPG-180
 - viewing GPG-271
- window clipping GPG-173, GPG-179
- window-clipping functions GPG-178
- with MakeScore TSD-33
- working with TSD-5
- working with real-time MIDI TSD-23
- Worksheet window
 - MPW DBG-9
- world axes GPG-242
- world coordinate system GPG-242, GPG-134
 - right-hand GPG-133
- world-space origin GPG-134
- wrap information
 - setting and getting GPG-168
- wrap modes
 - of TexBlend object GPG-161
- Wrapped file GSG-47
- Wrapper chunk GSG-46
- write permissions
 - memory fence restrictions of PPG-8
- WriteBackDCache() PPG-64
- WriteBattClock() PPG-172
- WriteChunk() PPG-254, PPG-255, PPG-269
- writetochunk DSG-36
- writemarkertable DSG-35
- WriteRawFile() PPG-147
- writestopchunk DSG-35, DSG-36
- writestreamheader DSG-15

X

- X rotation GPG-132
- X wrap mode (of TexBlend object) GPG-161
- XAxis GPG-248
- XER register DBG-44
- XSize attribute GPG-140
- xxx_Create() functions GPG-109

Y

- Y rotation GPG-132
- Y wrap mode (of TexBlend object) GPG-161
- yaw function GPG-244
- yaw functions GPG-244
- yaw operation GPG-246
- YAxis GPG-248
- Yield() PPG-4, PPG-22, PPG-24

yielding CPU time PPG-22
Yon attribute GPG-273
yon clipping plane GPG-272
 getting GPG-275
 setting GPG-274
YSize attribute GPG-140

Z

Z direction GPG-272
Z rotation GPG-132
ZAxis GPG-248
Z-axis
 negative GPG-272
Z-banding GPG-175
Z-buffer GPG-201, GPG-202, GPG-175,
GPG-176, GPG-182
z-buffer GPG-237
Z-buffer controls GPG-173
Z-buffer functions GPG-175
Z-buffering GPG-111, GPG-112, GPG-175,
GPG-176
ZCntl attribute GPG-175
Z-value
 of extended sprite objects GPG-88
Z-window GPG-176, GPG-177
ZXOffset attribute GPG-175
ZYOffset attribute GPG-175



3DO M2 Release Notes 2.0

Version 2.0 – May 1996

Copyright © 1996 The 3DO Company and its licensors.

All rights reserved. This material constitutes confidential and proprietary information of The 3DO Company. This documentation is subject to a license agreement with The 3DO Company and may be used only by parties to such agreement. Use by any other persons, and/or for any purpose not expressly authorized by the agreement, is strictly prohibited.

3DO's LICENSOR(S) MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. 3DO'S LICENSOR(S) DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME JURISDICTIONS. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

3DO and the 3DO logos are trademarks and/or registered trademarks of The 3DO Company. ©1996 The 3DO Company. All rights reserved. Other brand or product names are the trademarks or registered trademarks of their respective holders.

Table of Contents

Preface	RLN-xi
About This Document.....	RLN-xi
About This Release.....	RLN-xi
Audience.....	RLN-xi
Organization of This Document	RLN-xi
Typographical Conventions	RLN-xii
1.0 Introduction to 3DO M2 Release 2.0	RLN-1
1.1 READ THIS NOW!!!	RLN-1
1.2 Notes	RLN-1
1.3 Known Bugs and Caveats	RLN-2
2.0 Installing 3DO M2 Release 2.0	RLN-2
2.1 Notes	RLN-2
2.1.1 Important Points of Software Installation Process	RLN-2
2.1.2 On-Line Documentation	RLN-3
2.1.3 Post-installation Setup - Before Running the Debugger	RLN-3
3.0 Hardware.....	RLN-4
3.1 Known Bugs and Caveats	RLN-4
4.0 C Development Tools.....	RLN-5
4.1 READ THIS NOW!!!	RLN-5
4.2 Known Bugs and Caveats	RLN-5
4.3 Compiler	RLN-5
4.3.1 Notes	RLN-5
4.3.2 Known Bugs and Caveats	RLN-6
4.4 ANSI C Support	RLN-6
4.4.1 READ THIS NOW!!!	RLN-6
4.4.2 Notes	RLN-6
4.4.3 Fixed Bugs	RLN-6
4.5 Link3DO 1.15 & Dump3DO 1.0.4	RLN-6
4.5.1 Notes	RLN-6
4.6 3DO Debug 2.1a13	RLN-7
4.6.1 Notes	RLN-7
4.6.1.1 Tips and Tricks To Set Up Your Program For Debugging	RLN-7
4.6.2 Known Bugs and Caveats	RLN-11
4.7 Comm3DO Application 1.1a14	RLN-11
4.7.1 READ THIS NOW!!!	RLN-11
4.7.2 Notes	RLN-11
4.7.3 Known Bugs and Caveats	RLN-12
4.8 Logic Analyzer	RLN-12
4.9 CreateM2Make	RLN-12
4.9.1 READ THIS NOW!!!	RLN-12
4.9.2 Known Bugs and Caveats:	RLN-12
4.10 UserStartup•3DO	RLN-13
4.10.1 Notes	RLN-13
5.0 OS	RLN-13
5.1 Notes	RLN-13

5.2	Known Bugs and Caveats	RLN-14
5.3	Kernel	RLN-14
5.3.1	Notes	RLN-14
5.3.2	Known Bugs and Caveats	RLN-14
5.3.3	Fixed Bugs	RLN-15
5.4	Kernel - MemDebug	RLN-15
5.4.1	Notes	RLN-15
5.5	Device Drivers	RLN-15
5.5.1	Notes	RLN-15
5.5.2	Fixed Bugs	RLN-16
5.6	File System	RLN-16
5.6.1	Notes	RLN-16
5.6.2	Fixed Bugs	RLN-17
5.7	Event Broker	RLN-17
5.7.1	Notes	RLN-17
5.8	FSUtils Folio	RLN-17
5.8.1	Fixed Bugs	RLN-17
5.9	IFF Folio	RLN-17
5.9.1	Fixed Bugs	RLN-17
5.10	International Folio	RLN-18
5.10.1	Notes	RLN-18
5.11	New Folios	RLN-18
5.11.1	Notes	RLN-18
5.12	Math Calls	RLN-18
5.12.1	Notes	RLN-18
5.12.2	Fixed Bugs	RLN-18
5.13	Programs	RLN-18
5.13.1	Notes	RLN-18
6.0	Graphics	RLN-19
6.1	READ THIS NOW!!!	RLN-19
6.2	Known Bugs and Caveats	RLN-19
6.3	Graphics Folio	RLN-20
6.3.1	Notes	RLN-20
6.3.2	Known Bugs and Caveats	RLN-21
6.3.3	Fixed Bugs	RLN-21
6.4	2D API	RLN-22
6.4.1	Notes	RLN-22
6.4.2	Known Bugs and Caveats	RLN-23
6.5	3D Graphics Libraries	RLN-23
6.5.1	Notes	RLN-23
6.5.2	Framework	RLN-23
6.5.2.1	Known Bugs and Caveats	RLN-23
6.5.3	Pipeline	RLN-23
6.5.3.1	READ THIS NOW!!!	RLN-23
6.5.3.2	Known Bugs and Caveats	RLN-24
6.5.3.3	Fixed Bugs	RLN-24

6.6	Command List Toolkit (CLT)	RLN-24
6.6.1	Notes	RLN-24
6.6.2	Known Bugs and Caveats	RLN-25
6.6.3	Fixed Bugs	RLN-26
6.7	GState (graphics state)	RLN-26
6.7.1	Notes	RLN-26
6.8	Portable Binary SDF	RLN-26
6.8.1	Notes	RLN-26
6.9	Texture Settings	RLN-27
6.9.1	Notes	RLN-27
6.10	Texture Lib 1.0a9m2	RLN-27
6.10.1	Notes	RLN-27
6.10.2	Fixed Bugs	RLN-27
6.11	Texture Tools 2.0	RLN-28
6.11.1	Notes	RLN-28
6.11.2	Known Bugs and Caveats	RLN-28
6.11.3	Fixed Bugs	RLN-28
6.12	Geometry Compiler 1.0a6n	RLN-28
6.12.1	Notes	RLN-28
6.12.2	Known Bugs and Caveats	RLN-29
6.13	Modeling Package Converters	RLN-29
6.13.1	READ THIS NOW!!!	RLN-29
6.13.2	3DStudio 1.0a7 (3dstosdf)	RLN-29
6.13.2.1	Notes	RLN-29
6.13.2.2	Fixed Bugs	RLN-29
6.13.3	Strata 1.0a11 (ssptosdf)	RLN-29
6.13.3.1	Known Bugs and Caveats	RLN-29
6.13.3.2	Fixed Bugs	RLN-30
6.13.4	Lightwave Converters 1.1a2m1 (lwtosdf) & (lwstoanim)	RLN-30
6.13.4.1	Notes	RLN-30
6.13.4.2	Fixed Bugs	RLN-30
6.14	PostPro 2.0a3	RLN-30
6.14.1	READ THIS NOW!!!	RLN-30
6.14.2	Known Bugs and Caveats	RLN-31
7.0	Video and Data Streaming	RLN-31
7.1	READ THIS NOW!!!	RLN-31
7.2	Notes	RLN-32
7.3	Data Streaming run-time libraries	RLN-32
7.3.1	Data Stream run-time library	RLN-33
7.3.1.1	READ THIS NOW!!!	RLN-33
7.3.1.2	Notes	RLN-33
7.3.1.3	Known Bugs and Caveats	RLN-34
7.3.2	Subscriber library	RLN-34
7.3.2.1	READ THIS NOW!!!	RLN-34
7.3.2.2	SAudio Subscriber	RLN-35
	Known Bugs and Caveats	RLN-35

7.3.2.3	MPEG Video Subscriber	RLN-35
	Notes	RLN-35
7.3.2.4	MPEG Audio Subscriber	RLN-35
	READ THIS NOW!!!	RLN-35
7.3.2.5	EZFlix Subscriber	RLN-36
	READ THIS NOW!!!	RLN-36
	Notes	RLN-36
	Known Bugs and Caveats	RLN-36
7.3.2.6	DATA Subscriber	RLN-36
7.3.3	DS Utils library	RLN-37
7.3.3.1	dsblockfile.c	RLN-37
	Notes	RLN-37
7.3.3.2	mempool.c	RLN-37
7.3.3.3	msgutils.c	RLN-37
7.3.3.4	threadhelper.c	RLN-37
7.4	Data Streaming Examples	RLN-37
7.4.1	Notes	RLN-37
7.4.2	Fixed Bugs	RLN-37
7.4.3	PlaySA	RLN-38
7.4.3.1	Notes	RLN-38
7.4.3.2	Known Bugs and Caveats	RLN-38
7.4.3.3	Fixed Bugs	RLN-38
7.4.4	VideoPlayer	RLN-38
7.4.4.1	Notes	RLN-38
7.4.4.2	Known Bugs and Caveats	RLN-38
7.4.5	EZFlixPlayer	RLN-39
7.4.5.1	Notes	RLN-39
7.4.5.2	Known Bugs and Caveats	RLN-39
7.5	Data Streaming Tools	RLN-39
7.5.1	READ THIS NOW!!!	RLN-39
7.5.2	AudioChunkifier	RLN-40
7.5.2.1	READ THIS NOW!!!	RLN-40
7.5.2.2	Known Bugs and Caveats	RLN-40
7.5.3	SAudioTool	RLN-40
7.5.4	MPEGVideoChunkifier	RLN-41
7.5.4.1	READ THIS NOW!!!	RLN-41
7.5.4.2	Notes	RLN-41
7.5.4.3	Known Bugs and Caveats	RLN-42
7.5.5	MPEGAudioChunkifier	RLN-42
7.5.5.1	Known Bugs and Caveats	RLN-42
7.5.6	EZFlixChunkifier	RLN-42
7.5.6.1	READ THIS NOW!!!	RLN-42
7.5.6.2	Known Bugs and Caveats	RLN-43
7.5.7	QTVideoChunkifier	RLN-43
7.5.7.1	Notes	RLN-43
7.5.7.2	Known Bugs and Caveats	RLN-43

7.5.8	DATAChunkify -- the DATA Subscriber's Data Prep Tool	RLN-43
	Known Bugs and Caveats	RLN-43
7.5.9	Weaver	RLN-44
7.5.9.1	Notes	RLN-44
7.5.9.2	Known Bugs and Caveats	RLN-44
7.5.9.3	Fixed Bugs	RLN-45
7.5.10	Dumpstream	RLN-45
7.5.11	Worksheet and Exercises	RLN-45
7.5.11.1	READ THIS NOW!!!	RLN-45
7.6	Video Tools	RLN-45
7.6.1	3-2 Pulldown, MovieEdit, MovieCompress	RLN-45
7.6.1.1	Notes	RLN-45
7.6.1.2	Known Bugs and Caveats	RLN-46
7.6.2	EZFlux QT Component 1.0b1	RLN-46
7.6.2.1	Notes	RLN-46
7.6.3	Sparkle 2.4.5	RLN-46
7.6.3.1	Notes	RLN-46
7.6.4	SoundManager 3.1	RLN-46
8.0	Audio	RLN-46
8.1	READ THIS NOW!!!	RLN-46
8.2	Notes	RLN-47
8.3	Known Bugs and Caveats	RLN-47
8.4	Fixed Bugs	RLN-47
8.5	Beep Folio (new)	RLN-47
8.6	Audio folio	RLN-47
8.6.1	Notes	RLN-47
8.6.1.1	Envelope Time Scaling	RLN-47
8.6.1.2	Added Instrument query function family	RLN-48
8.6.1.3	Added other miscellaneous query support	RLN-48
8.6.1.4	Velocity Zones for MultiSamples	RLN-48
	Known Bugs and Caveats	RLN-48
8.6.1.5	Velocity Response Curves	RLN-48
8.6.1.6	Other Audio Folio Changes	RLN-49
8.7	DSP Instruments	RLN-49
8.7.1	Notes	RLN-49
8.8	Audio shell commands	RLN-49
8.8.1	audioavail (new)	RLN-49
8.8.2	dspfaders	RLN-49
8.8.3	insinfo (new)	RLN-49
8.8.4	makepatch	RLN-49
8.8.4.1	Notes	RLN-49
8.9	AudioPatch folio (new)	RLN-50
8.9.1	Patch compiler (CreatePatchTemplate())	RLN-50
8.9.1.1	Notes	RLN-50
8.9.2	PatchCmdBuilder	RLN-50
8.9.2.1	Notes	RLN-50

8.10 AudioPatchFile folio (new)	RLN-50
8.10.1 Notes	RLN-50
8.11 Music library	RLN-50
8.11.1 Patches	RLN-50
8.11.1.1 Notes	RLN-50
8.11.2 3DSound	RLN-50
8.11.2.1 Notes	RLN-50
8.11.3 Sample loader	RLN-51
8.11.3.1 Notes	RLN-51
8.11.4 Sound player	RLN-51
8.11.4.1 Notes	RLN-51
8.11.5 Sound spooler	RLN-51
8.11.5.1 Known Bugs and Caveats	RLN-51
8.12 Audio Examples	RLN-51
8.12.1 Notes	RLN-51
Appendix A Microcard	RLN-52
A.1 MicroCard Usage	RLN-52
A.2 How to Prolong the life of the MicroCard	RLN-52
Appendix B MPEG Read-This-First Roadmap	RLN-54
B.1 MPEG standards	RLN-54
B.2 MPEG Decoders	RLN-54
B.2.1 MPEG Video Device	RLN-54
B.2.1.1 READ THIS NOW!!!	RLN-54
B.2.1.2 Notes	RLN-54
B.2.1.3 Fixed Bugs	RLN-55
B.2.2 MPEG Audio Device	RLN-55
B.2.3 MPEG Audio Decoder DLL	RLN-55
B.2.3.1 READ THIS NOW!!!	RLN-55
B.3 3DO M2 support for real-time MPEG-1 decoding	RLN-56
B.3.1 Alternative to MPEG-1 Audio	RLN-56
B.4 Encoding/formatting MPEG streams for 3DO M2 playback	RLN-56
B.4.1 Interchange formats: MPEG-1 Video, MPEG-1 Audio, AIFF Audio ..	RLN-56
B.4.1.1 MPEG-1 Video format caveats	RLN-57
B.4.2 MPEG-1 Video encoding alternatives	RLN-58
B.5 A few words about MPEG-2	RLN-59
B.6 Additional Documentation	RLN-59
Appendix C Man Pages for MPEG Audio Functions	RLN-60
C.1 CreateMPAudioDecoder	RLN-60
C.2 DeleteMPAudioDecoder	RLN-60
C.3 MPACompressedBfrReadFn	RLN-61
C.4 MPAFlush	RLN-62
C.5 MPAGetCompressedBfrFn	RLN-62
C.6 MPAudioDecode	RLN-63
Appendix D Examples in Tools Folder.....	RLN-65
D.1 Audio Examples	RLN-65
D.1.1 test3sf	RLN-65

D.2	Development Code Examples	RLN-65
D.2.1	Comm3DOExampleClient 68K	RLN-65
D.2.2	Comm3DOExampleClient PPC	RLN-65
D.2.3	C3 Task	RLN-65
D.3	Graphics Examples	RLN-65
D.3.1	M2 Font Examples	RLN-65
D.3.2	M2 Kanji FontViewer	RLN-66
Appendix E	Examples in Examples Folder.....	RLN-67
E.1	READ THIS NOW!!!	RLN-67
E.2	Audio Examples	RLN-67
E.2.1	Beep	RLN-67
E.2.2	Juggler	RLN-67
E.2.3	Misc	RLN-67
E.2.4	Score	RLN-68
E.2.5	Sound3D	RLN-68
E.2.6	SoundEffects	RLN-69
E.2.7	Spooling	RLN-69
E.3	Event Broker Examples	RLN-69
E.4	FileSystem Examples	RLN-70
E.5	Graphics Examples	RLN-70
E.5.1	CLT	RLN-70
E.5.2	Fonts	RLN-71
8.12.1.1	READ THIS NOW!!!	RLN-71
8.12.1.2	Fonts Examples	RLN-71
E.5.3	Frame	RLN-71
E.5.4	Frame2d	RLN-72
E.5.5	GraphicsFolio	RLN-72
E.6	Kernel Examples	RLN-72
E.7	Miscellaneous Examples	RLN-73
E.8	MPEG Examples	RLN-73
E.9	Data Streaming Examples	RLN-73
Appendix F	Selected Man Pages for Examples	RLN-75
F.1	anim	RLN-75
F.2	CLTspinclip	RLN-75
F.3	CLTsurface	RLN-75
F.4	freespun	RLN-76
F.5	GSquare	RLN-76
F.6	newview	RLN-76
F.7	playsf	RLN-77
F.8	scrolling	RLN-77
F.9	SeeSound	RLN-78
F.10	ta_bee3d	RLN-79
F.11	ta_leslie	RLN-81
F.12	ta_sound3d	RLN-82
F.13	ta_steer3d	RLN-83

Preface

About This Document

This document describes the technical changes and bug fixes for the 2.0 release of the 3DO M2 tools and operating system.

Some components include their own release notes on the distribution media in the form of ReadMe files. Besides feature highlights, these component-specific release notes also contain important caveats and warnings. Take the time to carefully read through them as there are a few key “important notes” that will make the difference between your code running or not.

About This Release

This 3DO M2 2.0 release is an update to the 1.2 release and provides a number of enhancements.

This release includes the following items:

- M2_2.0 Software CD
- Read This First
- 3DO M2 Release Notes, Version 2.0.
- 3DO M2 Developer's Documentation Set

Note: *These Release Notes appear in hardcopy and in ascii format on the CDROM. In addition, there are component-specific release notes on the CDROM with the individual components. The hard copy Release Notes take priority over the on-line release notes. Differences between the hardcopy Release Notes and the on-line Release Notes are indicated by change bars in the hardcopy Release Notes.*

Audience

This document is for developers preparing titles for the M2 Development System.

Organization of This Document

- **Preface**
Brief description of this document.
- **Introduction to 3DO M2 Release 2.0**
Notes about the release as a whole. This section contains some or all of the following subsections:
 - **Introductory paragraph(s).**
Brief overview or description.
 - **READ THIS NOW!!!**
Particularly urgent or important information about the overall release.

- Notes
Notes about the overall release.
- Known Bugs and Caveats
Known bugs and things to watch out for that are of particular importance to the overall release.
- Fixed bugs
Fixed bugs that are of particular importance to the overall release.
- **Installing 3DO M2 Release 2.0**
Notes on installing the release. This section contains some or all of the following subsections concerning the installation process:
 - Introductory paragraph(s).
 - READ THIS NOW!!!
 - Notes
 - Known Bugs and Caveats
 - Fixed bugs
- **Sections on individual components of the 3DO M2 system** (e.g., Hardware, C Development Tools, Graphics, Video and Data Streaming, Audio).
Each section contains subsections about subcomponents. Each section or subsection contains some or all of the following subsections relating to the overall component or individual subcomponent:
 - Introductory paragraph(s).
 - READ THIS NOW!!!
 - Notes
 - Known Bugs and Caveats
 - Fixed bugs
- **Appendices**
Additional information.

Typographical Conventions

The following typographical conventions are used in this book:

Item	Example
code example	<code>Scene_GetStatic(scene)</code>
procedure name	<code>Char_TotalTransform()</code>

Item	Example
new term or emphasis	In M2, <i>characters</i> are objects that can be displayed on the screen.
file or folder name	It is located in " <i>folder:subfolder</i> ".

1.0 Introduction to 3DO M2 Release 2.0

This Release 2.0 of the 3DO M2 Portfolio and Toolkit consists of new, updated, and improved hardware, software, and documentation.

The following sections contain information about the release as a whole. Information about individual components of the M2 system are contained in subsequent sections.

1.1 READ THIS NOW!!!

- **M2 Release 2.0 runs on Rev G or later dev cards. It does not run on Rev E or earlier dev cards.**

- **Setting the Current Directory**

The current directory of a program is now set by the system to refer to the directory where the executable being run is located. That is, the current directory of a program may be different than the current directory of the shell used to invoke the program.

This behavior is intended to make it easier to write programs that don't rely on knowledge of their location. All of the files for a program can be accessed relative to the program's initial current directory, since it is assured that the current directory will always point to where the application is.

This behavior can be defeated by specifying the `-Hflags=1` command-line argument when linking your programs. This causes the linker to set a bit in the executable that tells the system to initially set that program's current directory to be the same as that of the shell it was launched from. This behavior is what a normal shell environment typically does, and is desirable for any shell utility programs like "ls" for example.

- **Compatibility Concerns**

Many changes in this release can cause compatibility problems in your title. Please review each section carefully if you suspect any compatibility-related issues.

- **Font Folio rewritten.**

The Font Folio was rewritten to improve performance. Significant changes were made to the API. See the *3DO M2 Graphic Programmer's Reference* for details.

- **Encryption**

A title may not be submitted for encryption with the 3DO M2 Portfolio and Toolkit release 2.0, but will be possible with a future release.

- **Documentation**

With this release of the 3DO M2 Portfolio and Toolkit release 2.0 software, there is a completely updated set of documentation. If you have received documentation before this release, you are advised to replace it all with the new documents provided.

1.2 Notes

- The reference documentation has been substantially expanded and updated.

1.3 Known Bugs and Caveats

- Please refer to the Graphics Programmer's Guide for the most up to date information on Graphics. The man pages may be out of date or incorrect. This warning does not hold true for the 2D API or the Graphics Folio.
- The following could cause compiles to fail:
 - ioi_Unit has been removed from the IOInfo structure. References to this can be safely removed.
 - <mathf.h> no longer exists. References to this should be changed to math.h.
 - <mpegvideo.h> is new and replaces <mpegdevice.h>.
 - • MAXINT was defined in values.h which no longer exists. Use INT_MAX in limits.h.
- An application that uses the serial port or audio-input can be tested only under the debugger. The serial port and audio-input will not work properly when used by an application booted from a CD-ROM or cdrom.image file.

2.0 Installing 3DO M2 Release 2.0

This section provides important installation instructions that you should follow when installing this 3DO M2 Portfolio and Toolkit software CD, release 2.0 content on to your development system. Currently, the supported development platform is the Apple Macintosh.

2.1 Notes

2.1.1 Important Points of Software Installation Process

The important points of the software installation process are listed below. Detailed installation instructions for both the hardware and software are covered in the document, "Getting Started With 3DO M2 Release 2.0" included with your documentation.

- Drag the file UserStartup•3DO from the CD to your MPW folder

The installer records where the 3DO software was placed; the new UserStartup•3DO file uses that location to set the MPW environment variables.

Note: *If you rename your old version of UserStartup•3DO, make sure that the new name doesn't begin with "UserStartup." By convention, MPW executes all files that begin with UserStartup. Executing two or more copies of UserStartup•3DO will result in an unusable development environment.*

- Run the installer program

Note: *Note: The progress bar does not currently work.*

- Install additional folders as needed

In an attempt to reduce the disk space requirements for an installation, the following folders are not installed. If the folders had MPW tools, the tools can be found in {3DOFolder}tools:M2_2.0:MPWTools: after installation.

Folders that are not installed can be found on the CDROM.

```
/* tools:M2_2.0:AS_IS */
/* Examples:M2_2.0:Streaming:Tools_And_Data */
/* tools:M2_2.0:Audio:/remote/ARIA/samples V1.3 */
/* tools:M2_2.0:Audio:3SF */
/* tools:M2_2.0:Audio:AudioThing */
/* tools:M2_2.0:Audio:DumpMF */
/* tools:M2_2.0:Audio:PIMap Utilities */
/* tools:M2_2.0:Audio:Sound Manager 3.1 */
/* tools:M2_2.0:Audio:SquashSnd */
/* tools:M2_2.0:Graphics:Conversion Tools */
/* tools:M2_2.0:Graphics:M2 Kanji FontViewer */
/* tools:M2_2.0:Graphics:ShareWare Fonts */
/* tools:M2_2.0:Graphics:Texture Tools */
/* tools:M2_2.0:LowLevel:dump3do */
/* tools:M2_2.0:LowLevel:link3do */
/* tools:M2_2.0:LowLevel:M2MacCompression */
/* tools:M2_2.0:LowLevel:PPCAS */
/* tools:M2_2.0:Video:EZFlix QT */
/* 3do_os:M2_2.0:remote:Examples:Audio:MarkovMusic:Data */
```

Independent Release Notes that can be found with the Tools in their individual folders are also not installed; those release notes have more detailed information in many cases than what's noted later in these Release Notes.

- Delete the following file from your system folder:
System Folder:Extensions:3DO:3DO_Setup

2.1.2 On-Line Documentation

On-line documentation is covered in "Getting Started With 3DO M2 Release 2.0" included with your documentation.

When first accessing PortfolioHelp you will receive a dialog that says there is no index and would you like to create one. Click OK. After MPW creates the index you may receive the following errors:

```
### duplicate key - Box3_ExtendPt
### duplicate key - Pt3_Transform
### duplicate key - ls
### duplicate key - type
### duplicate key - walker
```

You may safely ignore these warnings.

2.1.3 Post-installation Setup - Before Running the Debugger

To run the debugger on an 8 Mb dev card:

1. Copy the file "DEVROM.m2.flash.unenc.str" into your dev card's Flash ROM. You must use the FlashRomTool to do this.
 - Run the **FlashRomTool**.
 - In the window titled "Onboard Flash", click in the Filename box.
 - Select the file "DEVROM.m2.flash.unenc.str".
 - Click the Flash button.
 - When you see the "Verified" message, quit the FlashRomTool.

You only need to do this once. You do not need to perform this first step again unless you receive a new OS release from 3DO.

2. In your release folder, remove the file named "dbinfo.8M" and then rename the file "dbinfo.devel.8M" to "dbinfo.8M".
3. Set up the debugger.
 - Run the debugger. You must use debugger version 2.1a10 or later.
 - Under the Target menu, select Setup.
 - Make sure the ROM is set to "Debugger".
 - Click on the Script button and select the file "debugger.flash.scr".

To run the debugger on a **16 Mb dev card**:

1. In your release folder, remove the file named "dbinfo.16M" and then rename the file "dbinfo.devel.16M" to "dbinfo.16M".
2. Set up the debugger.
 - Run the debugger. You must use debugger version 2.1a10 or later.
 - Under the Target menu, select "Setup".
 - Make sure the ROM is set to "Debugger".
 - Click on the Script button and select the file "debugger.16M.scr".

3.0 Hardware

3.1 Known Bugs and Caveats

- The following bug is known to occur with all development cards and could cause visual artifacts noticeable to developers using this development system.
LOD determination can be incorrect for thin triangles.

In very thin (narrow) triangles, some spans will not have any pixels drawn on them, making the lines appear dashed. This is expected. The problem is that the vertical LOD logic is not ignoring these lines for the purposes of generating VLodLatch and VLodCompare signals. So a VLodCompare signal is being generated to compare against a previous line's VLodLatch, except that previous line had no pixels in it.

The result is that the LOD determination made by the Texture Mapper based on horizontal and vertical u,v deltas may be incorrect for these triangles.

- When a texture is displayed at extreme perspective (as on the side of a cube), the image exhibits blockiness in the vertical direction. This means that you see visible banding when looking at a vertical column of pixels, while a horizontal row of pixels looks fine. This problem is apparent on the spinning cube demo, especially with the mandrill and Cindy textures.

The cause has to do with the LOD determination performed by the Triangle Engine. The triangle engine looks at the texel delta in both the horizontal and vertical direction, and chooses the coarser LOD. In the case described above, there is significant compression of the texture in the horizontal direction, but little compression in the vertical direction. This means that the LOD is determined by the horizontal delta, with the effect that, vertically, a coarser LOD is used than is desired. This means that several successive vertical pixels

will sample the same texel in the coarser LOD, resulting in blockiness in the vertical direction.

The same effect may be seen at the top of the cube, if the top face is just barely visible, but this time the vertical compression drives the LOD choice, and the blockiness appears in the horizontal direction.

There is no way to avoid these problems, but generally when triangles are so thin as to cause this to occur, the texture will already be so warped (such as on the side of a barely visible cube) that the overall degradation of the image should be slight.

4.0 C Development Tools

4.1 READ THIS NOW!!!

Due to changes in the executable file format, all executables compiled and linked with older versions of `dcc` and `link3do` must be relinked with the new linker or they will not run with this release. For more details, see the Kernel section later in this document.

4.2 Known Bugs and Caveats

The tasker IDs of Mac build Examples are the name of the makefile that generated them (appname = <makefile name>). Consequently, any makefile that produces multiple output files produces Examples that all have the same ID to the debugger. This makes killing a task within the debugger difficult.

This is especially serious for the EventBroker Examples, which will all have the ID "eventbroker". The Debugger will malfunction when `killtask` is attempted because EventBroker is a critical part of the system.

You can bypass this problem by using `CreateM2Make` on each individual source file.

4.3 Compiler

4.3.1 Notes

- The compiler has been upgraded to revision 3.7a. This version continues to support source-level debugging as well as fixes a number of bugs. For more information see the Diab 3.7 compiler release notes included with the documentation.
- The Diab compilers are now shipped fat. This should speed up compiles and builds on Power Macs.
- The default makefiles that are included in the "Examples" folder use the diab default assembler "das", and do not take advantage of the faster "ppcas" assembler.

However, the default makefiles do place a copy of the executable file inside the "Remote:Examples:..." folder relevant to the program. For example, "Newview" is placed in

"3DO:3DO_os:M2_2.0:Remote:Graphics:Frame:Newview".

This can be useful for certain examples.

A new makefile, created by the command "`CreateM2Make`", will use `ppcas` by default, but the executable will be placed in the

"3DO:3DO_os:M2_2.0:Remote" folder because CreateM2Make only contains generic directory variables for its output.

If you want the output file to be in its "Remote:Examples:..." folder, you must move it (along with its accompanying ".spt" file for source-level debugging) to that location. However, leaving the executable file in the ":Remote" folder, could make directory navigation easier while running the debugger.

4.3.2 Known Bugs and Caveats

- If you abort a compilation using Command-., the temporary asm files in "{TempFolder}" are not deleted. You must manually delete these files to avoid filling up your disk.

Note that "{TempFolder}" is the folder "RAM Disk:temp:" if that folder exists. Otherwise, "{TempFolder}" is "{Boot}System Folder:Preferences:MPW:TempMPW:".

4.4 ANSI C Support

4.4.1 READ THIS NOW!!!

Strict ANSI checking does not allow C++ style comments "/*". To allow such comments, do not specify the dcc flag -Xstrict-ansi.

4.4.2 Notes

- Removed non-functioning stdio code and replaced it with error messages warning that the functions aren't currently supported, and pointing developers to adequate substitute functions. This affects `fopen()`, `fclose()`, `fread()`, `fseek()`, `ftell()`, and `fgets()`.
- `putc()`, `fputc()`, `puts()`, `fputs()`, and `fwrite()` now always send their output to the debugging terminal, since file I/O is not currently supported.
- Added `cprintf()` and `vcprintf()` which are similar to `sprintf()` and `vsprintf()`, except that instead of putting characters into a buffer, a callback is invoked for every character generated by the formatting process.
- Implemented `strcpy()` differently, which makes Diab generate code that's twice as fast.

4.4.3 Fixed Bugs

- Fixed `system()` to correctly return the result code of the program being run.

4.5 Link3DO 1.15 & Dump3DO 1.0.4

4.5.1 Notes

This release includes a new version of *link3do* and *dump3do*. Changes since the 1.2 release are noted below.

- Link3do 1.15 on TK 2.0 is now a Fat Binary. For developers using link3DO 1.15 on a PPC the MPW stack size must be adjusted. This is very important as otherwise your Mac will crash. To set stack size, use the MP "setshellsize" command. Modify the -s argument to 128K or greater, then reselect the "setshellsize" command and hit "enter." Quit MPW and restart in order for the change to take effect.

-
- The executable file format has been changed. All executable files are now 3DOELF compliant. KOFF is no longer supported. The command `fixup3do` is no longer used to complete the linker phase of creating an executable file.
 - All executables must be relinked with this linker due to changes in executable file format.
 - If you have project make files that built a M2 application under M2_1.2 or earlier, you will need to update those make files with new link command lines because of the migration of some libraries into DLL (modules). Examples of correct link command lines can be found in the Examples folder (see "Examples in Examples Folder" on page 67) or you can generate a basic make file with `CreateM2Make` that ships with this release.
 - To allow your compiled object files to work with the new linker, make sure your makefiles include the compiler options
`-Xsmall-data=0 -Xsmall-const=0.`

4.6 3DO Debug 2.1a13

The debugger has been updated to version 2.1a13.

4.6.1 Notes

- This debugger provides all features of the debugger included in the previous release as well adding support for a development platform-resident file system. This allows reading and writing of files on the development platform from code running on the 3DO M2 system. This debugger includes source-level debugging.
- If you are planning to use the source-level debugging capabilities of 3DODebug there are several compiler and linker flags you should be aware of:
 - The `-g` option tells the compiler to generate source-level debugging information in the object file. The symbol information and the executable are now contained in a single file.
 - Using the `-g` option (debugging) in conjunction with optimized code will probably not execute precisely as expected and some lines may be optimized out completely (in which case, there will be no breakpoint available for that line.)
 - If you're using the `-g` option, we recommend you remove the following optimizations:
`-XO -Xunroll=1 -Xtest-at-bottom -Xinline=5` and then set `-Xinline=0`
 - The `-Xinline=0` option tells the compiler that it should not optimize functions into inline code. Allowing the compiler to optimize functions in this manner can be very confusing if you try to set breakpoints in functions that are converted to inline code.
 - `-XO -Xunroll=1 -Xtest-at-bottom -Xinline=5` will produce optimized code.
- For Source Level Debugging, the application heap size (of 3DODebug) should be at least 4 MB plus the size of the executable.

4.6.1.1 Tips and Tricks To Set Up Your Program For Debugging

- Hints for debugging:
 - Make sure that compiler optimizations
`-XO -Xunroll=1 -Xtest-at-bottom -Xinline=5`

are removed from the makefile.

- Make sure that option `-g` is used for source debugging.
- Make sure the `-k` or `-g` option is used on the `link3do` command line if the objects were not compiled with `-g` to preserve symbols.
- Make sure the symbol file to which the debugger is pointing is the symbol file being loaded and run or an exact copy of it. When you update your OS, make sure that the Data directory points to the current location of your symbol file.
- For source-level debugging, the source directories must be set up correctly before the symbol file is read.

Try the following if the debugger is not working properly:

- Remove the `"3DODebug.prefs"` file from the debug file. This will force you to reset your directory paths, which may be incorrect.

If that does not work, try removing the `"3DO Directory Prefs"` file from the `"Preferences"` folder in the System Folder.

- Rebuild the entire application, including all objects and libraries.
- If you drop into Macsbug with the message

`Out of memory, ES and increase application partition.`

or you get a dialog box with the message

`The application "unknown" has unexpectedly quit because an error for of type -491 occurred.`

then you have run out of memory and must increase 3DODebug's memory partition.

- If you are accessing files remotely, then try copying the files to the local machine.

Known problems:

- FTP fails to transfer files correctly for files larger than 400K. Try copying the file again using a different transfer scheme.
- Debugging across the network seems unreliable

Hints for linking

- The files specified on the link line must be specified in the following order: OBJECT FILES then DLL MODULES then LIBS. Any other order is not guaranteed to work and may cause the linker to issue incorrect messages and crash.
- If building a C++ example, make sure you have a `libd.a` library and are linking with it.
- If the linker complains about an invalid alignment, use the `-Ax` option to increase the alignment to a value greater than that of any sections within the objects (32 or 64 should be enough).
- Rebuild all files (including libraries) and watch for any warnings emitted by the compiler.

If you are experiencing problems, try the following:

- Increase the memory partition of MPW shell in the finder's `"Get Info"` window

-
- Carefully check all linker options, paying special attention to whether referenced libraries really exist.
 - For unresolved symbols, repeat the libraries on the command line.
 - Be sure you are not using the -m or the -m2 options. You can avoid this problem and the previous one by using CreateM2Make to generate your makefile.
 - The Link line has been changed due to the addition of a new library. For 2.0 and beyond, the clt library must be on the link line BEFORE the graphics lib. CreateM2Make has been updated to reflect this change, but for any makefiles which you do not plan to rebuild, you should at least apply this change. See the Graphics Section of this document for further clarification and an example.
 - The link line has been changed to support DLLs. Several libraries have been removed and replaced by DLL's.

Compiler

- Treat any warnings as errors.
- If compiling C++, don't declare static objects.
- C++ debugging is not supported. If you're trying to debug C++, the debugger tries to ignore any strange C++ dwarf tags. Depending on the structure of the tag, this may not work.

If the program isn't able to run or compile, try the following:

- Remove optimizations and see if that helps.
- If -g is used, remove it.
- Try putting `printfs` (or external calls) in problem code - this can sometimes help the compiler produce correct code.

Problems

- If you populate a character array with a constant string of more than 928 characters (which is over the ANSI maximum of 512) and try to compile your program, the compiler will crash. Subsequent attempts to compile this or any other program will fail.

Workaround: Shorten the offending variable, save the file, restart the computer, and then recompile the program.

Narrowing down a test case:

- Find the problem object.
- Edit the object and start removing code until the code is small but still causes problems.
- Dump the object using `dump3do` to see if anything about the file looks questionable.
- The include file `<kernel : debug . h>` defines handy macros for printing debugging information with compile-time-controllable verbosity. The verbosity level can be controlled on a file-by-file basis.
- `Debug.h` also defines macros for error checking, reporting, and handling. These macros don't suffer from the nasty problems of the Opera `debug3do.h` error reporting macros: they never call `exit()`. The ones that

handle error recovery do so at all verbosity levels; they compile to consistent C syntax (void expression or {block}) at all verbosity levels, and they won't cause compiler warnings at different verbosity levels.

It's important to control the verbosity because printf's slow down program execution potentially disrupting streaming and other real-time activities. Excess printf's can also bury the important ones.

The first thing to know about is the macro `DEBUG`. Use

```
#ifdef DEBUG
```

or

```
#if defined(DEBUG) && SOME_OTHER_SWITCH
```

to conditionally compile debugging code. Use the `-DDEBUG` compiler switch to enable debugging code in a particular source file.

- ANSI C hint: If there's no macro `XXX` defined, the compiler acts as if the macro was defined as 0. Some compilers warn about `"#if XXX"` when `XXX` is undefined. Diab C doesn't.
- The macro `APRNT()` always compiles in a debug printf. In a development OS, the printf will come out to the debug Terminal. In a production OS, the printf will come out to the serial port, if available.
- The argument to `APRNT()` is a parenthesized argument list for printf, for example:

```
APRNT((" MPEG Video PTS deltas ranged [%d to %d] "
      "with frames ready [%d to %d] audio ticks early.\n",
      debugStats->minDeltaPTS, debugStats->maxDeltaPTS,
      debugStats->earliestFrame, debugStats->latestFrame));
```

- ANSI C hint: You can break a long string constant into multiple lines, as in this example.
- The macro `APERR()` works just like `APRNT()`, but it's used for error messages. It helps to distinguish error printouts from informational printouts in source code. E.g. you can grep for `APERR` or change `APERR()` calls to `PERR()` calls.
- The next batch of macros are all verbosity-controlled by compile-time policy switches `DEBUG` and `PRINT_LEVEL`. You can set these switches on the compiler's command line, e.g. `-DPRINT_LEVEL=2`. Or your source code could set them **before** it includes `<kernel:debug.h>`.

`defined(DEBUG) || PRINT_LEVEL >= 2` -> print all messages.

`PRINT_LEVEL == 1` -> print error messages.

`PRINT_LEVEL == 0` or undefined -> print none of these messages. This is the default.

- The macros `PERR()` and `PRNT()` are like `APERR()` and `APRNT()` with verbosity control.
- The macro `ERROR_RESULT_STATUS(s, res)` prints an error message string `s` and a system result code `res` (using `PrintfSysErr`), with verbosity control. For example:

```
ERROR_RESULT_STATUS(
    InitMPEGWriteBuffer FMVCreateIOReq", status);
```

-
- The macro `PRINT_RESULT_STATUS(s, res)` acts like `ERROR_RESULT_STATUS(s, res)`, at print-level verbosity.
 - The macro `ERROR_NIL_STATUS(s)` prints an error message about a NULL pointer expression, with the identifying string `s`. `PRINT_NIL_STATUS(s)` is the print-level verbosity version.
 - The macro `CHECK_NEG(s, res)` checks for a negative result code `res`. If `res` is negative, it prints the string `s` and the result code `res` via `ERROR_RESULT_STATUS(s, res)`. Use `CHECK_NEG(s, res)` to check and report an error status when you don't need the macro to help with error handling. (See also `FAIL_NEG`, below.)
 - The final batch of macros help with error handling in addition to error reporting.
 - `FAIL_NEG(s, res)` checks if result code `res` is negative. If so, it uses `ERROR_RESULT_STATUS(s, res)` to report the error and then does "goto FAILED".

4.6.2 Known Bugs and Caveats

- In the Target Setup dialog, the check box item "Initial bp in boot code" should never be selected. This functionality is broken, and will cause your target system to hang during boot.
- In the Source window, the PC arrow will only show up if it is at the first instruction of a source statement. Because of a bug in the line information, the PC arrow will occasionally disappear when source stepping. This most commonly occurs when stepping through while and for loops.
- Source files larger than 32K will not display correctly in the Source window.
- The debugger has trouble switching between development cards. If you have two (or more) cards in your Mac, you can switch between cards by choosing "setup" from the "target" menu. A dialog will appear. The topmost pulldown menu is the "target." At this point, you can switch cards. Then click "OK" in the dialog and you will notice the message "M2 released from reset" in the Terminal window of the debugger. At this point, you must quit the debugger and re-run the debugger.
- 3DODebug will report a syntax error when trying to display a variable name that is also the name of a field in a structure.

4.7 Comm3DO Application 1.1a14

Comm3DO is an architecture that allows two-way communication between an application running under the Macintosh OS (a "double-clickable" application) and a task or program running under the 3DO portfolio.

4.7.1 READ THIS NOW!!!

Comm3DO App cannot be run simultaneously with the Debugger.

4.7.2 Notes

Using Comm3DO, a Macintosh application ("client") can identify target development cards, boot the 3DO M2 portfolio, launch and kill tasks, and enjoy two-way communications with Comm3DO-savvy tasks. More than one client application can talk to a development card simultaneously. Communications

sockets are allocated dynamically to a client upon request: 128 unit numbers are reserved for this purpose, any one of which can be used either exclusively or on a shared basis.

- What do you get?

The pieces are:

- the Comm3DO App
- the Comm3DO.h interface, for Macintosh client applications
- the Comm3DO_portfolio.h interface, for Comm3DO-savvy tasks running on M2.
- the Comm3DO Example Macintosh Client,
- the source code of a Comm3DO-savvy task.

This communications package and sample application allow host-resident tools to run code on the development platform allowing results to be displayed on the target development card. Examples of this use are found in PostPro, which is also included in this distribution.

4.7.3 Known Bugs and Caveats

- Owing to last-minute changes in the use of the fields of the IoInfo structure (kernel:io.h) for the host device, the portfolio-side source code included with Comm3DO on the M2_2.0 CD for Mac-to-Portfolio communications is incorrect. As a result, tasks built with this code, including the Comm3DO example task included on the CD, will not work.

Workaround:

1. Discard the file c3task in the ":Comm3DO:Example Client Task:" folder.
2. Open the file Comm3DO_portfolio.c in the ":Comm3DO:interfaces:libraries:" folder.
3. Replace every occurrence of "ioi_unit" in that file with "ioi_CmdOptions".
4. Set the MPW directory to ":Comm3DO:Example Client Task:" and rebuild c3task.
5. Certain makefiles, including Comm3DO when trying to compile the Comm3DO client task, have problems using Link3DO. The workaround is to remove the -m2 option from the link3do line in the makefile.

4.8 Logic Analyzer

The Logic Analyzer card is not available with this release.

4.9 CreateM2Make

4.9.1 READ THIS NOW!!!

Developers are expected to create new makefiles with each release of the software CDs.

4.9.2 Known Bugs and Caveats:

- Don't create any makefiles that request optimization and debugging to be turned on simultaneously.

-
- Some compiler options have been removed, because they are now stored in the config file.
 - The makefile now creates a <> .spt file along with the executable. This script tells 3DO Debug where to find source code and symbols. Note that this does not work with multiple source directories. See the 3DO Debugger manual for more information.
 - On a PowerPC Macintosh, attempting to view source directories if they are set using a ".spt" file will crash the Macintosh with an illegal instruction error. Note that CreateM2Make automatically generates a ".spt" file.

This is not a problem if you are using a 68k Macintosh or if you are viewing the directory when it is set manually.

4.10 UserStartup•3DO

4.10.1 Notes

- Has been modified to work with the new compiler version 3.7a.
- UserStartup•3DO may cause your builds to run improperly if you have spaces in your path names.
- UserStartup•3DO creates a menu item to allow copying of user items to a RAM disk.

Note that copying libraries to a RAM disk (a menu item is provided to do this) does not significantly decrease development time and can require as much as 5 MB of RAM. Space taken up by the libraries (and header files) will not be available for temp files written by the Diab compiler during compilation.

Using a RAM disk to store temp files is effective and does not require checking a menu item. If you use a RAM disk for this purpose, you should make certain that the size of the RAM disk is greater than the size of any individual temp file that may be written there.

Even if your RAM disk is larger than any single temp file, there is a possibility that several temp files may persist on the RAM disk instead of being removed immediately after use. If this happens, compiler errors of various kinds may result. The Diab compiler will not automatically remove temp files to make room for new ones. The workaround is to manually remove these orphaned files before attempting a new compile.

5.0 OS

This section details changes in operating system components between release 1.2 and 2.0. Changes to the graphics, audio, and streaming software are not included.

5.1 Notes

- Major work has been done to fix known bugs and address known deficiencies throughout the system.
- This release includes some reduction in OS memory usage. More to come in the future.
- Ongoing work to reduce OS CPU overhead continues. Many performance enhancements are included in this release, and more will follow.

- This release includes much behind-the-scenes work to prepare the system for being put into ROM and onto a CD.
- This release includes support for 16M development systems.
- This release includes support for CD-ROM image files, which lets you build a complete CD image file and run it through the debugger as if it were a real CD.
- Changed the type of the “wait” parameter in `GetControlPad()` and `GetMouse()` to be “bool” instead of “uint32”, since that’s what they really are.
- Considerably speeded up the `FindLSB()` function.
- Added the `libdebugconsole.a` library. This provides a set of functions to open a debugging view console on the 3DO display where `printf()`-style output can be sent. Given the large time overhead in doing a `printf()` to the host, the local debugging view console is a much better solution for somewhat real-time printouts.

5.2 Known Bugs and Caveats

- The CD-ROM drive is currently only run at 2x instead of 4x due to a hardware problem.
- The debugging console view must currently always be opened full-screen due to a hardware bug.

5.3 Kernel

5.3.1 Notes

- Task wake up from `WaitPort()`, `WaitIO()`, and `LockSemaphore()` is now considerably faster which improves overall system performance.
- MMU handling has been optimized to minimize task switching overhead, especially when swapping in and out of the various system daemons.
- Added development-time output to `DeleteItem()`. Whenever `DeleteItem()` fails, it prints out warning messages explaining the failure. This will help track down resource leaks where the client thinks an item is being deleted, but it is failing for some reason (wrong owner for example).
- Added the `ReadUniqueID()` function and the `<kernel:uniqueid>` header file that gives access to the machine ID hardware register.
- Added the `CreateIODebug()` which replaces the functionality that used to be offered by the `IOStress` program.
- Added support for persistent memory (memory that can remain present across reboots). See `ControlMem()` for details.
- Removed the `ioi_Unit` field from the `IOInfo` structure in `<kernel/io.h>`, since the kernel no longer supports unit numbers.
- Tweaked some kernel headers so they can be used from C++.

5.3.2 Known Bugs and Caveats

- The `OpenModule()` function will fail if you pass in a NULL filename.
- The `ImportByAddress()` function does not work in the 2.0 release of the operating system. Use `ImportByName()` instead.
- The loader example programs are not included on the 2.0 CD.

5.3.3 Fixed Bugs

- Fixed many bugs in the executable loader and DLL manager.
- Fixed task killing code which wasn't always registering the correct task as the murdering task.
- Fixed task launch code so that argv[argc] is now set to NULL on start-up, to conform to ANSI C.
- Fixed bug in ControlMem() where attempting to prevent all tasks from writing to a memory block wouldn't actually work.
- Fixed bug in FlushDCacheAll() which was trashing a register it shouldn't have been.
- Fixed bug where a task's old priority was being recorded in Lumberjack when changing a task's priority, instead of recording its new priority.
- Fixed several problems in the debugger monitor, which fixes the debugger's Stop feature and a number of other problems.
- Fixed bug where IsMemReadable() wasn't returning ROM addresses as readable.
- Fixed handling of item closing when a task dies or when it closes a folio. It was possible for certain items to get skipped over and never get closed.
- Corrected the definition of MEMTYPE_ILLEGAL in <kernel/mem.h>.
- Fixed OpenDeviceStack() to clean up on error.
- Made ReadHardwareRandomNumber() work on BDA 2.0 systems.

5.4 Kernel - MemDebug

5.4.1 Notes

- DumpMemDebug() by default now only displays information about the current task. The new tag DUMPMEMDEBUG_TAG_TASK lets an arbitrary task or all tasks be included. The new tag DUMPMEMDEBUG_TAG_SUPER controls whether allocations made from supervisor mode are included in the output.
- SanityCheckMemDebug() now takes an extra banner argument that is used to identify which call to SanityCheckMemDebug() is triggering an error within a program. Without this parameter, it was sometimes almost impossible to figure out where a memory block was getting corrupted.
- Added RationMemDebug() which provides control over a new vicious feature of the memory debugging system. You can now cause selected memory allocations to fail. This is very useful to test failure paths in code. By using the many tags, you can specify that one out of N allocations be rationed, that only allocations within a given size range fail, that allocations should be rationed randomly over a given interval, etc.
- When displaying information about a corrupted memory region, MemDebug now displays the actual address of the memory block.

5.5 Device Drivers

5.5.1 Notes

- Totally new implementation of the microslot driver. The old driver used to consume nearly 100% of the CPU time when reading or writing to a

microcard, and this version consumes somewhere in the range of 5% of the CPU time. In addition, the old driver used to do about 10 task switches per second, this new one does none. Finally, this driver should be generally more reliable.

- Performance improvements, latency reductions, cleanups, bug fixes, and other tweaks in the control port device. This includes turning on 4x transfer speed for reduced response time.
- Much cleanup and finalization in the 16550 serial driver. This includes making the driver work on production systems, some performance enhancements, and some bug fixes.

5.5.2 Fixed Bugs

- Fixed a few bugs in the storagecard driver, including support for reads and writes of more than 64K.
- Fixed a few bugs in the CD-ROM driver, mainly related to redbook CD handling.

5.6 File System

5.6.1 Notes

- Cleaned up and consolidated the code which handles filesystem remounts, and duplicate filesystem names. Filesystems of all types are now automatically renamed at mount time, if their name conflicts with that of a previously-mounted filesystem.
- File systems are now dynamically loaded and unloaded in order to reduce memory consumption.
- Added the Rename() system call. This currently only works for microcards and not on /remote. This call lets you rename files, directories, and file systems.
- File system aliases are now global instead of task-private.
- Added the ability to prevent the FindFileAndXXX() routines from searching specific types of filesystems.
- The MountFileSystem() call no longer has a unit parameter, it instead takes a device item and block offset.
- The FILECMD_FSSTAT I/O command no longer returns a unit number in the status data structure. Instead, it now returns the Item number of the underlying device stack.
- Added flag bits to the MinimizeFileSystem() call to force filesystems to be quiesced, force their driver code to be unloaded, and force filesystems to be dismounted (the latter is `_dangerous_`)!
- Added the FileSystemMountedOn() call, which will scan the list of mounted filesystems and return the Item number of the first filesystem which is mounted on the DeviceStack specified by the caller.
- Set the FS attribute of the RAM, microcard-ROM, and microcard drivers to appropriate levels. This will have the effect of deferring the mounting of the RAM and microcard devices until after the shell has started up and had a chance to run the system start-up script (if one is present).
- Completed implementation and fixed many bugs in file system mounting and dismounting.

5.6.2 Fixed Bugs

- Fixed a few bugs in the Acrobat file system (used on microcards).
- Fixed a crash when sending FILECMD_FSSTAT to a file opened on /.
- Fixed a bug in OpenDirectoryPath() where it didn't cleanup after itself in case of failure. It was forgetting to close an opened file.

5.7 Event Broker

5.7.1 Notes

- Changed the DeviceOnline and DeviceOffline events to a single DeviceChanged event.
- In development mode, the Event Broker now checks to make sure that the "reserved for future use, must be zero" fields in the configuration message are indeed zero.
- The event broker now checks that messages requesting information are buffered messages.
- Added preliminary support for the new MEI M2 Control Pad.
- Added the ControlSettingsChanged event which is sent if any Control Pad analog control changes value.

5.8 FSUtils Folio

5.8.1 Fixed Bugs

- Fixed a bug in the file copier engine where it wasn't detecting errors that could occur when first allocating the blocks for new files.
- Fixed a bug in the file copier engine that caused it not to delete partial files that could not be fully written out because of an error.
- Removed excessive unneeded semaphore locking within the file copier engine. This eliminates a considerable number of system calls when doing file copies.
- Fixed a bug where the file copier engine would terminate prematurely if it transitioned from reading to writing while scanning the top level directory during a copy operation. This situation happens when there isn't enough memory in the system to hold all the files in the top level directory.
- Fixed bug where the copier could end up in an infinite loop if there was not enough memory to allocate a single file or directory node.
- Fixed bug in the copier where it wouldn't call CloseDirectory() on one directory entry if a copy operation was aborted for some reason. This caused a memory and item leak.
- Fixed bug in the file deleter engine where it didn't stop deleting when the progress callback told it to.

5.9 IFF Folio

5.9.1 Fixed Bugs

- Fixed a bug where the CollectionChunk.cc_Container field wasn't being set to the right thing during a parse operation, causing clients to crash when using the value.

- Fixed a bug where SeekChunk() with IFF_SEEK_CURRENT wasn't correctly limiting negative seek offsets.
- The SeekChunk() prototype now returns an Err instead of an int32.
- SeekChunk() no longer ignores attempts to seek past end of files.

5.10 International Folio

5.10.1 Notes

- Updated to get the language from the battery-backed memory if possible.

5.11 New Folios

5.11.1 Notes

- Finalized design and implementation of the Icon folio.
- Added the SaveGame folio, which provides a very easy way to read and write saved game files to storage, and have the data be automatically compressed and decompressed.
- Added the Requester folio, which provides a GUI to let the user navigate and organize file systems.
- Added the Batt folio which deals with the battery-backed clock and battery-backed 15 bytes of memory.
- Added the Date folio which provides date conversion primitives which are useful in conjunction with the batt folio.

5.12 Math Calls

5.12.1 Notes

- Added new very fast math routines: rsqrtfff(), rsqrtff(), rsqrtf(), sqrtfff(), and sqrtff().
- Made sqrtf() 5 times faster, one third the size, and more accurate.
- Sped up and reduced size of fabsf(), floorf(), ceilf(), ldexpf(), frexpf(), sinf(), cosf(), modff(), and fmod().
- The double-precision math calls, such as sin() and cos(), are not available on the M2 machine. Use single-precision calls, such as sinf() and cosf(), instead.
- Added inline assembly for some functions.
- <mathf.h>, <math1f.h>, and <values.h> have been removed from the external distribution.

5.12.2 Fixed Bugs

- Fixed bugs in itof() and utof() and improved their performance considerably. These routines are used by the compiler to convert between integer and FP formats.

5.13 Programs

5.13.1 Notes

- Replaced the EraseCard program with the new more general WriteMedia program.

- Added the System.m2/Programs/Clock program which lets you view and adjust the time and date of the battery-backed clock.
- Updated the Items program to deal with the latest systems structures and device command definitions.
- Revised the System.m2/Programs/Intl program to set the system language in the battery-backed memory using the Batt folio.
- Several filesystem-related programs, such as Mount and AcroFormat, no longer take a unit parameter on the command-line.

6.0 Graphics

6.1 READ THIS NOW!!!

- When linking a graphics program, the following link line should be used:
`-lmusic -lframe2d -lframe_utils -lframework -lbsdf -lpbsdf -lframework -lpipeline -lframe_utils -lbsdf -lpbsdf -lclt -lspmath -lfile -leventbroker -lc`

- Compressed textures are not supported by the Revision E development card. While working with this card you should create all textures as uncompressed and use the Texture library to compress them afterwards, if you wish. The same texture API can be used to render both compressed and uncompressed textures.

Compressed textures are supported by Revision G development card.

- Minimizing Footprint: to decrease your memory footprint, keep in mind the following:
 - use indexed textures whenever possible
 - make sure that textures, materials and models are shared wherever possible.
- Interfacing with C++

The Graphics header files are designed for inclusion in C++ programs. In order to include the Graphics header files from C++ you should make sure that the preprocessor variable `GFX_C_Bind` is defined in your makefile.

6.2 Known Bugs and Caveats

- The font example makefiles ("`fonts.make`" and "`showmessage.make`") located in the directory:
`"M2_2.0:3DO:Examples:M2_2.0:Graphics:Fonts:"` do not work. They are backlevel versions and will terminate during the build.
 However, all the source files in that directory are valid and can be compiled using makefiles generated with `CreateM2Make`.
- The instructions to run the "anim" program that are included in the header of the source file "`anim.c`" are out of date.

To run anim, do the following:

- Build the anim program using the default makefile or a makefile created by `CreateM2Make`.

If you are using a new makefile created by `CreateM2Make`, move your "anim" executable (and ".spt" file) into the following folder:

`3DO:3DO_os:M2_2.0:Remote:Examples:Graphics:Frame:Anim`

The `skeleton.csf` file is already in the directory by default

You can also leave the `anim` program where the `CreateM2Make` makefile placed it, and move or copy the `skeleton.csf` file needed to run the example into the same directory.

- Launch 3DODebug and navigate to the directory containing the `anim` executable.
- At the command line, type
`anim skeleton.csf skeleton_world`

The usage message that appears if you enter the command incorrectly is also out of date.

The correct usage is:

`anim <filename>.csf <filename>_world`

6.3 Graphics Folio

6.3.1 Notes

- Projectors

The biggest change to the Graphics folio is the addition of the "Projectors" facility, which supports differing video modes, such as NTSC and PAL. To support this new facility, some substantial changes have been made to the internal workings of the folio. Most of these changes are invisible to your applications.

The major changes are described below:

- There is a new "device driver", called "bdavideo.privdevice", which contains the Projector data structures and support code. This driver is loaded and initialized by the graphics folio when the graphics folio is loaded.
- The graphics folio now manages a new Item, called a "Projector." A Projector can be thought of as the root ViewList for a given video mode.
- There is a new data structure, called a "ViewTypeInfo." Applications can inspect this structure to discover important characteristics of the type of View in use, such as maximum possible pixel dimensions.
- The View Item itself has two new fields:
`v_ViewTypeInfo` -- pointer to this View's ViewTypeInfo structure.
`v_Projector` -- pointer to the Projector Item handling this View type.
- Substantial documentation has been written about Projectors, both in the man pages (reference manual pages) and in the expository chapter in the manual.
- For those of you that read the man pages, you may have noted that I've been promising for some time to completely re-do the whole View Type numbering scheme. The first hints at how this will look are now present in `<graphics/view.h>`.

They do not actually work yet! But they should give you an idea where things will be going. When the View Type numbers do change, compatibility code will be put in place to assure all your existing code will still run.

- There is a new example program showing how to automatically resize Views to the prevailing video mode. It is in
"`examples/graphics/graphicsfolio/autosizeview.c`".

- The graphics folio now informs the system boot process that it has assumed control of the video hardware.
- The graphics folio contains support for BDA 2.1 chips.
- Views are now properly centered horizontally for a given video encoder.
- Virtually all of the folio man pages have been updated, as well as the expository chapter in the manual.
- Notes about the Triangle Engine Device Driver
 - A new command has been added to the Triangle Engine device driver: `GFXCMD_SETVIEWBITMAP`. This command will, upon completion of all prior TE I/O commands, perform a Bitmap buffer swap for you.
 - The Triangle Engine device driver now recovers from lockups better.
 - The driver commands now have man pages.
 - Z-buffers may now be rendered to as ordinary, 16-bit Bitmaps, making certain Z-buffered effects easier.
 - Turned on the Triangle Engine's 16-byte burst write ability, enabling it to write small triangles more efficiently.

- **Optimum Z-buffer Alignment**

3DO M2 hardware has the ability to access "alternate" 4K pages of memory simultaneously (e.g. an even- and odd-numbered page could be accessed at the same time). An ideal situation to take advantage of this is when rendering Z-buffered images; the triangle engine can write the pixel and Z-buffer value in one memory cycle. Until now, however, achieving that alignment was difficult and caused you to waste memory.

The system supports "co-alignment" of Bitmap buffers. If you have a Z-buffer you wish to optimally align with an existing Bitmap, you may request co-alignment. The system will then fill in the `bm_BufMemCareBits` and `bm_BufMemStateBits` fields to indicate the state the bits in the buffer pointer must have to achieve optimal alignment. `AllocMemMasked()` can be used to obtain memory meeting these requirements.

Use Tag `BMTAG_COALIGN` to specify the Item number of the Bitmap against which you wish to co-align your new Bitmap.

6.3.2 Known Bugs and Caveats

- With release 2.0, there is a small performance degradation with double buffering. The workaround when performing benchmarking is to use single buffering. This will be fixed in the next release.

6.3.3 Fixed Bugs

- Earlier versions of the BDA silicon had a bug that caused video lines that had interpolation (pixel averaging) enabled to be shifted one 640-wide pixel to the right. Version 1.2 of the graphics folio compensated for this, but did not uncompensate for later versions (BDA 2.0 and later), when the bug was fixed. This has been fixed.
- Fixed a bug where, if you attempted to create a View without a Bitmap, the folio would crash the machine.
- Patched bad interaction between Triangle Engine device driver and graphics folio which could cause `RenderSignals` to be dispatched one field too early.

6.4 2D API

6.4.1 Notes

- GridObj support has been enabled in 2.0. A GridObj is a structure that has a pointer to a rectangular 2 dimensional array of sprite objects, and also sets a position and dimensions for the drawing of the grid. The following routines have been enabled for GridObj support (see the man pages for details):

Gro_Create

Gro_Delete

Gro_SetSpriteArray

Gro_SetWidth

Gro_SetHeight

Gro_SetPosition

Gro_SetHDelta

Gro_SetVDelta

Gro_GetSpriteArray

Gro_GetWidth

Gro_GetHeight

Gro_GetPosition

Gro_GetHDelta

Gro_GetVDelta

- The function `Spr_CreateShort()` has been added, which creates a short sprite object (a sprite object that has no geometry). Short sprite objects can be used to set texture or destination blend attributes without causing anything to be drawn, or they can be used with the GridObj structure.
- As of release 2.0, sprite objects now correctly support textures that are larger than 16K. There may be slight differences in the actual rendering between Revision E development card systems and Revision G development card systems due to limitations in the hardware support before Revision G development cards.
- Added sprite command list caching to enhance sprite performance.
- Added a feature to enhance performance for sprites that are known to be always completely on the display: the routine `Spr_DisableClipping()` bypasses logic in the `F2_Draw()` function to clip the sprite against the edges of the display.
`Spr_EnableClipping()` can be used to turn on the clip code for the sprite. Sprites that have clipping disabled will produce unexpected results if they overlap the edges of the display.
- Added new routines to make loading sprites from texture files easier. See the man pages for the routines `Spr_LoadTexture()`, `Spr_LoadTextureVA()`, and `Spr_UnloadTexture()`.

6.4.2 Known Bugs and Caveats

- There is a bug in the 2D sprite code. If a sprite's texture is changed (through `Spr_SetTextureData()` or `Spr_CreateTxData()`), subsequent rendering of the sprite may still use the old texture that was previously associated with the sprite.

This bug will be fixed in a subsequent release.

6.5 3D Graphics Libraries

There are two 3D Graphics Libraries: the Pipeline and the Framework. The Framework is an object-oriented, extensible API designed specifically for building and manipulating three-dimensional scenes and sits on top of the Pipeline.

The Pipeline is an optimized rendering library in the 3DO M2 3D graphics system. For further information on these pieces, please refer to the *Graphics Programmer's Guide* in the documentation set.

6.5.1 Notes

- The 3D library now uses GState for its command list management. This allows users to share command list buffers between the 2D graphics, 3D graphics, and Font Folio.
- Changed `testspin` so it now defaults back to a reasonably large size for the objects.

6.5.2 Framework

The 3DO M2 Graphics Framework is an object-oriented, extensible API that is designed specifically for building and manipulating three-dimensional scenes. In the overall 3DO M2 architecture, the Framework is situated between your application and the Graphics Pipeline.

6.5.2.1 Known Bugs and Caveats

- If you are using the built in shapes: cube, cylinder, torus, sphere in an SDF file, the primitive will receive the wrong material index. This does not affect the SDF files created by our conversion tools, since they never use these primitives.
- If you cannot close a binary SDF file, you should recompile your code with this 2.0 release.
- `Scene_SetTransparent` is not working correctly. Currently, setting scene transparency to `FALSE` turns it on, while setting it to `TRUE` turns it off. The default is still to have scene transparency off.
- Support for orthographic projections has been removed from the API in this release.

6.5.3 Pipeline

The 3DO M2 Graphics Pipeline, or GP, is an optimized rendering library in the 3DO M2 3D graphics system. In M2, the Graphics Pipeline is situated between the Graphics Folio and the Framework.

6.5.3.1 READ THIS NOW!!!

- Only one GP is allowed per application.

6.5.3.2 Known Bugs and Caveats

- You cannot use the scaling transforms with lighting, this will lead to an incorrectly illuminated object.
- There is a bug in the 3d pipeline which causes the hardware to hang when rendering faceted pre lit triangle surfaces.

A temporary buffer is allocated for holding the transformed and clipped vertices, initially this is 32k (each vertex can occupy up to 36 bytes). At the start of each primitive the space needed to hold the vertices is compared against the current size and if necessary the buffer is reallocated, however no check is done in the 2.0 version of the software to ensure that this allocation was successful. If the system is running low on memory and the primitive is too large to fit in the current buffer then the machine will hang.

- The only time when a possibly valid return code would be interpreted as TXB_None (-1) is in the case where the user gets the DbIDitherMatrixA and DbIDitherMatrixB. Values of DitherMatrix are only valid if you actually set them. The user should ignore the non-error code.
- If you are using a macro in the Graphics Lib, do not treat it as a function; a macro to a pointer with ++ will lead to unforeseen results.

6.5.3.3 Fixed Bugs

- The bug that caused a crash when all lights in the scene were disabled has been fixed.

6.6 Command List Toolkit (CLT)

The Command List Toolkit is a collection of routines and defines which allow the user to build command lists easily and efficiently for use by the triangle engine. The CLT was created in order to allow developers to easily talk directly to the Triangle Engine through the Device Driver. By providing this API, developers can easily port other 3D engines to M2.

6.6.1 Notes

- The macros which used to reside in "<:graphics:clt:clt.h>" have been moved to "<:graphics:clt:cltmacros.h>", which is included in "<:graphics:clt:clt.h>".

This should be transparent to your applications because the new folder is included in the old one.

- CLT_TXTIPCNLT and CLA_TXTIPCNLT now cause correct output to the command list. See the man pages for the proper ordering of arguments.
- The CLT_TXTTABCNT() macro has been renamed to CLT_TXTTABCNTL(). This was a typo in the 1.1 release.
- The CLA_TRIANGLE() and CLT_TRIANGLE() macros now take an additional parameter, "new". This "new" is a Boolean flag stating whether this triangle instruction should start a new triangle strip/fan, or re-use two vertices from the last strip/fan. In earlier releases, this parameter was always implicitly set to 1.
- The CLT texture loader now supports compressed textures and carved textures, on Revision G development cards.
- Man pages are now available for utility routines provided by libclt.a

- Tips and Tricks for Debugging CLT and Command Lists
 - 3DODebug has a Triangle Engine disassembly window in it.
 - The Triangle Engine ignores the most significant 8 bits of addresses. The result of this is that when you look for data such as SrcBaseAddress, etc., you will see a value of the form 0x00xxxxxx. In our system's memory map, this address should be 0x40xxxxxx. For example, if the Destination buffer's address appears as 0x0010d580, the real address in memory will be 0x4010d580.
 - The memory-mapped registers are located at 0x40000 - 0x4FFFF. This state, however, will reflect the state at the last Sync, Pause or Triangle Engine error only. To see why a certain object is not rendering correctly, you can insert a Pause instruction into the list just after the triangles for that object are rendered, then do a while(1) in your code. This will give you time to examine the state of the Triangle Engine by opening a Data window in the Debugger.
 - On Revision E cards, do not attempt to examine the Triangle Engine registers while the T.E. might be busy. With the Revision E development card, this can cause the T.E. to hang. It is best to either exit the program, or put a while(1) in your code before you attempt to examine the data at 0x40000.
 - If you ever see a message which looks something like:

```
### T.E. Timeout failure, IRP = 0x4010D010
```

 that means that when the Triangle Engine crashed when it was reading from the address at IRP (Instruction Read Pointer). Usually, this does not mean that the data at the IRP address was bad. Because of the pipelined nature of the T.E., many parts of the hardware are actually executing instructions a few words above where the IRP is pointing at. Look backward in the list, starting at the IRP, to see if any data appears to be invalid.
 - The most common reason to get a T.E. timeout is that a Sync instruction was not in the command list after a collection of vertex instructions. Whenever you start sending one or more tristrip/trifan instructions, you must use a SYNC instruction before setting any new state for the Texture Mapper, Destination Blender, or other units of the Triangle Engine.
 - Negative X or Y values for vertices will also cause either a T.E. crash, or garbage all over the screen.
 - The first time a Triangle Disassembly window is opened after launching 3DODebug, it will attempt to obtain the address of the current command list by examining the register at 0x00040008.
 - If you are using the Destination Blender to blend with a source frame buffer, make sure that you have set the SrcBaseAddr register, and all of the other registers related to the source frame buffer.

6.6.2 Known Bugs and Caveats

- There is a bug when using the Triangle Engine under 2.0. The TE is only reset the first time the triangle engine is used. This means that the first program you run that uses the TE will be running with the TE in a clean state. Subsequent programs will start with the TE left in the state of the last program executed. This means that you must reset all the TE registers to known values or risk unpredictable results.

6.6.3 Fixed Bugs

- The functions `CLT_TXTPIPCNTL()` and `CLA_TXTPIPCNTL()` now cause correct output to the command list. See the man pages for the proper order in which to specify the arguments.
- All known typos in the CLT macros have been corrected. This eliminates some compile errors that occurred previously.

6.7 GState (graphics state)

6.7.1 Notes

- GState has now been moved out of `libclt.a`, and made into `gstate.dll`. Like other DLLs, GState will be loaded the first time your program requires one of its routines.
- GState no longer maintains the full array of bitmaps. As of the 2.0 release, it is only necessary to inform GState of the current frame buffer (will change each frame in double/triple-buffering schemes), and Z-buffer (normally won't change per frame). User code is now required to manage its own array/list of bitmaps. A utility routine, `GS_AllocBitmaps()`, is provided to easily allocate multiple bitmaps and place them in an array. For applications using the 3D Pipeline/Framework that are calling `InitGP()` or `GfxUtil_SetupGraphics()`, this bitmap Item array is maintained as a global in `PortInit.c` called `gBitmaps`.
- `GS_SetDestBuffer()` and `GS_SetZBuffer()` now take the bitmap's Item as a parameter. `GS_GetDestBuffer()` and `GS_GetZBuffer()` now return an Item as their result. This was done so that the validity of the bitmaps can be more easily checked when `GS_SendList()` is called.
- `GS_SetSrcToCurrentDest()` has been renamed to `CLT_SetSrcToCurrentDest()`. The arguments and functionality are unchanged.
- `GS_ClearFrameBuffer()` has been renamed to `CLT_ClearFrameBuffer()`. In addition, the new routine takes two additional Boolean arguments specifying whether to clear the frame buffer, and whether to clear the Z-buffer. Code being brought forward from earlier releases should simply add `", TRUE, TRUE"` to the end of the arg list.
- Man pages have been updated for other minor changes to GState routines. The published API for GState should now remain quite stable.

6.8 Portable Binary SDF

6.8.1 Notes

- This release includes a new binary SDF format, the primary goals of this format are:
 - To provide a simple interchange format between the 3d tools and systems software for the description of hierarchical 3d models and scenes. It should be simple enough for external developers to create this format from their existing models.
 - Software independence: the current binary file format is essentially a memory dump of the framework and pipeline data structures. This means that the binary format changes with each new release of the software.

-
- Platform Independence: the new data file format allows the data to be produced on any platform; UNIX, Mac or PC. It is the responsibility of the portable binary SDF Writer to produce the data in M2 byte order and floating point representation. (The PBSDF Writer is functionality that is provided in the Geometry Compiler tool.)
 - The code for portable BSDF can be found in Sources/Graphics/PBSDF.

6.9 Texture Settings

A fully updated and complete description of texture settings is provided in the manual, "3DO M2 Graphics Programmer's Guide" in this release.

6.9.1 Notes

- The graphics tools attempt to provide reasonable defaults for textures. However, if you are hand-editing SDF files be sure to check the following information:
 - All tiled textures must be a power of two (in each dimension).
 - All non-normalized texture coordinates should be in the range [0,1023].
 - You can not slice and dice tiled textures; therefore, make sure they do not exceed 16 KB in size (the Geometry Compiler (gcomp) warns about this.)

6.10 Texture Lib 1.0a9m2

6.10.1 Notes

- TAB and DAB manipulation functions have been exposed to the user and the headers are in the M2 TXattr.h file. For TAB and DAB attributes, see the Command List Toolkit documentation.
- All I/O functions will set proper file type and creator when run on the Mac. The file type will be set to "3DOt" (for a UTF texture) and the creator will be "Rean" which is Post Pro.

6.10.2 Fixed Bugs

- All previous versions of the libraries handled "transparent" formats (formats with the IsTrans flag set) incorrectly. Any compressed data should be regenerated with this new library or with tools compiled with this new library. Due to a bug in the library, all previous versions of tools would deal with "transparent" formats (formats with the IsTrans flag set) incorrectly. This means that all compressed data should be regenerated with new versions of these tools or custom tools compiled with this release of the 3DO M2 Texture Library. As always, compressed textures will only work with Revision G development cards.
- M2TX_Init function incorrectly set the default DCI constants to be White/Opaque/SSB OFF, it now sets them to be Black/Opaque/SSB OFF which should be the default.
- M2TX_UncomprToM2TXRaw and M2TX_ComprToM2TXRaw improperly handled the case of converting an indexed image. This has been fixed.
- Version 1.0a5 had a bug that prevented the Mac version to overwrite existing files, this bug has been fixed.
- M2TXRaw_MakePIP had a bug that allowed only the first call in a session to be successful. Subsequent calls in a session would give erroneous output. This has been fixed.

- `M2TX_Print` now prints out the number of colors present in a PIP (not just the size).

6.11 Texture Tools 2.0

6.11.1 Notes

- A new tool, `quanttopip`, has been added. This tool takes in an input image, a reference image with a PIP and will generate an output image that is a quantized version of the input image using the PIP of the reference image. This is useful for batch processing large numbers of images to make them share the same PIP.
- PPM texture tools are out of date. They have been archived for this release. If you would like more up to date Public Domain Texture Tools, check out the following public ftp servers:
 - `nic.funet.fi`
 - `ee.lbl.gov`

6.11.2 Known Bugs and Caveats

- When entering the name `ppmtoutf` to get usage info on the Macintosh or the SGI, you will not get a prompt back. You will need to interrupt the command, on the Mac (command period) and SGI (control c), and there will be no further problem.

6.11.3 Fixed Bugs

- Due to a bug in the library, Mac versions of the tools in the previous release could not overwrite existing files. This has been fixed.
- Due to a bug in the library, `utfmakesame` would generate bad data in the case someone wanted to convert an indexed image to a literal image. This has been fixed.
- Also, error checking has been relaxed to allow the decomposition of indexed images into literal components so that a person could extract an alpha or ssb channel that has been "folded" into the PIP.
- The previous version of `utfit` had a bug that caused it to miscalculate the size that an image should be to fit into TRAM in some cases. Also, `utfit` would resample an image even if it was already at the correct size to fit into TRAM. This could effectively cause an image to be resampled as many as three times in a batch processing script like `mippify`. This caused the images to become excessively blurry, leading to poor looking MIP-maps. `utfit` now only resamples an image if it is absolutely necessary.
- `utfinfo` now gives the number of colors present in the PIP.

6.12 Geometry Compiler 1.0a6n

6.12.1 Notes

- There is a new option for generating pre-computed lighting from an ASCII SDF input file that is only valid on the SGI.
- Updated to write portable binary SDF format. This format is based on "Mod 4 IFF" specification. By default this tool generates files in the new portable (IFF) binary format.
- Updated the portable binary SDF to output user extensions

-
- Removed support for ASCII SDF output (formerly option “-a”).

The default options are now:

1. Generate Portable Binary SDF
2. Use compiled surfaces
3. Decompose geometry into “snake” primitives. The options “-i”, “-o”, and “-s” will (respectively) disable each of these default options.

6.12.2 Known Bugs and Caveats

- The Geometry Compiler has problems dealing with large files.
- The Macintosh version of the Geometry Compiler behaves unreliably in some environments and on some data files.

6.13 Modeling Package Converters

There are different versions of these programs for different platforms so that they can convert the native file format of their modeling packages to SDF. Further details of the SDF format are described in the manual “3DO M2 Graphics Tools,” which is part of the 3DO M2 Developer’s Documentation Set.

6.13.1 READ THIS NOW!!!

- It is important that models converted with the 1.1 release be reconverted with the 2.0 release. The binaries have changed with this release, and will change again with future releases.

6.13.2 3DStudio 1.0a7 (3dstosdf)

6.13.2.1 Notes

- Writes unique names for an animation array. This is done by prefixing filename, for example: “skeleton_kfengines”
- The default Texblend information sets the blending between material and texture (txBlendOp Mult).
- Writes out “3DS smoothing groups”. 3DS smoothing groups are broken into separate TriMeshes so that it is possible to get sharp edges between different groups.

6.13.2.2 Fixed Bugs

- Now “DUMMY” nodes in 3D Studio are converted as empty nodes in the hierarchy.
- All string names in the SDF file are enclosed in quotes to fix the problem where string names start with numbers.
- In material, the “shadeenable” flag is written out.

6.13.3 Strata 1.0a11 (ssptosdf)

6.13.3.1 Known Bugs and Caveats

- There is a bug in Strata’s texture mapping dialogue; it shows up when you use default textures. You can set horizontal and vertical field %’s, but the number may or may not be output. The work around is to go back and retype a % in the field a second time. Change it and hit ok; otherwise you get a zero value and you will get weird texture coordinate number values or your software will crash. These will be fixed in the next release of the software.

6.13.3.2 Fixed Bugs

- Using ssptosdf with the -p option will no longer crash your Macintosh.

6.13.4 Lightwave Converters 1.1a2m1 (lwtsdf) & (lwstanim)

6.13.4.1 Notes

- There are two programs for Lightwave conversion: lwtsdf and lwstanim.
 - lwtsdf will convert a Lightwave object file (.lwo) to an ASCII sdf file.
Models will retain their names so a file called ship.lwo will be referenced in the SDF file as "ship.lwo".
Texture names are converted to lowercase, path names stripped away and given a .utf extension. At the end of the object processing, a list of textures and their new names is output in script form. This list can be output to a file to use as a batch processing script for the model's textures.
 - lwstanim will take a Lightwave scene file (.lws) and create an object hierarchy and a set of KeyFrame objects in the same format as those output by the 3D Studio and Strata converters.
The top-level node of the scene will be named "world" and can be referenced with an SDF viewer (like newview) to see the scene as it would be set up at time 0. To see the animation, the anim viewer must be used.
- A new option, -t, has been added to the converters. Using the -t option will suppress the output of texture when the geometry is being written out to the ASCII SDF file.
- Error messages have been added to notify you when an error has occurred during the writing out of ASCII SDF files rather than just exiting without any message.
- Path names are now stripped off the input files so that if for example path:test.lwo is given as an input file, material and textures will be test.lwo_materials rather than path:test.lwo_materials. This was to avoid excessively long names when people would use an absolute path (often through an environment variable) for input file names.
- lwstanim now will name the top level node "<output filename>_world" rather than just "world" so that multiple animations can be merged and compiled into a single file.
- Also the animation data structure name has been changed from "kfengines" to "<output filename>_kfengines" for the same reason. Note that the 2.0 Release of the anim program will be required to play these animations as older version will only look for "kfengines".

6.13.4.2 Fixed Bugs

- The previous bug with using -ext option on the Mac has been fixed.

6.14 PostPro 2.0a3

This is the second M2-specific PostPro release.

6.14.1 READ THIS NOW!!!

This version does not allow preview of 3D models on the 3DO M2 hardware.

6.14.2 Known Bugs and Caveats

- PostPro is only writing the first level of detail when doing a conversion from texture to RGB.
- Texture files with multiple LODs will be shifted to the right by about a quarter inch when saved to a new name from PostPro.
- If PostPro's memory is low, conversions from a texture may result in garbled images. It doesn't take much to max out the PostPro's memory. Therefore, if you are getting garbled images, try increasing PostPro's partition size.

7.0 Video and Data Streaming

"Data Streaming" is a general term which can refer to any or all of the following areas of software included in this release:

- Data Streaming run-time libraries

There are three application-linkable libraries that manage real-time streaming of data from CD.

Data types supported by these libraries include MPEG video, MPEG audio, EZFlx video, AIFF audio, and general data.

These libraries support MPEG playback from CD. Note that applications with very small video stills or short clips that can reside entirely in 3DO M2 RAM may wish to use the MPEG driver interface directly.

- Data Streaming examples

There are four example applications, which show how to use the Data Streaming API and demonstrate some of M2's capabilities, such as MPEG.

- Data Streaming tools

These are MPW tools for converting, preparing, and multiplexing standard audio, video, and other data formats into a real-time M2-playable stream.

For a thorough discussion on Data Streaming, please refer to the following documentation:

- 3DO M2 DataStreamer Programmer's Guide

Explains the "theory of operation" of Data Streaming.

- 3DO M2 DataStreamer Programmer's Reference

Provides reference information about Data Streaming API calls and data-preparation tools.

This information is also available in HTML pages and man pages.

7.1 READ THIS NOW!!!

- This release will not play streams containing MPEG Video created for older releases. You must use the new MPEGVideoChunkifier tool and the updated Weaver to rechunkify and reweave your data streams before playing them on an M2 Release 2.0 system.
- Because the audio clock is generated by the CD-ROM clock, the CD-ROM power must be on (and the red indicator light on) in order for audio output, audio timers, and Data Streaming to work. If a program is stuck, try turning on the CD.

7.2 Notes

- Data Streaming now supports branching by means of application calls to `DSGoMarker()` and GOTO chunks in the stream.
You can choose whether to flush the subscribers on each branch. A flush branch starts displaying new data as soon as possible at the cost of freezing playback during the seek. A non-flush branch aims to “cover” the seek by finishing out the queued data before playing the new data.
GOTO chunks normally perform non-flush branches, while `DSGoMarker()` normally performs a flush-branch. You should use these defaults in almost all cases.
“VideoPlayer” on page 38 and “EZFlixPlayer” on page 39 describe how to demonstrate branching.
- Your Weaver script specifies the size and number of stream buffers.
Small buffers let the streamer branch quicker (because branch time includes the time to read the first buffer), but larger buffers increase usable bandwidth (because less of the stream is spent on FILL chunks).
The total buffer space needed (size multiplied by the number of stream buffers) depends on the subscribers you’re using. Total buffer space also controls how long the streamer can “cover” a seek.
- There are two new subscribers, the MPEG Audio Subscriber and the DATA Subscriber, along with their chunkifier tools and example applications.
- The MPEG Video stream chunk format changed to improve its support of branching. You must re-chunkify your MPEG Video elementary streams to use them with the 2.0 version of the MPEG video subscriber.
- The “MPEGChunkifier” tool was replaced with separate `MPEGVideoChunkifier` and `MPEGAudioChunkifier` tools.
- Many “man pages” were added and improved (e.g., new man pages for all the stream development tools, Weaver script commands, audiochunkifier supported compression types, `DSGoMarker()` options, and `DSWaitEndOfStream()` `msg_Result` codes).
Man Pages generate HTML, MPW 411, and hardcopy (Data Streamer Programmer's Reference).
- The Data Streamer Programmer's Guide has been substantially updated.
- A number of bugs were fixed. The software is more thorough about bounds-checking channel numbers.

7.3 Data Streaming run-time libraries

This software, often referred to as the “Data Streamer,” is a set of three libraries that can be linked with a 3DO M2 application for the purpose of managing the following closely-related real-time tasks:

- Streaming timestamped (video, audio, etc.) data off of CD.
- Decompressing the data for presentation.
- Interfacing to the MPEG decoder at a higher level than the device drivers.
- Maintaining audio/video synchronization.
- Delivering any kind of data to the application in synch with audio and video.
- Branching, pausing, and resuming playback.

-
- Managing buffers for the above tasks.

The Data Streaming run-time modules are currently organized into a set of three linkable libraries:

- The Data Stream library: `libds.a`
- The Subscriber library: `libsubscriber.a`
- DS Utils: `libdsutils.a`

In addition, programs that use the EZFlixSubscriber will need to link in the EZFlix Decoder library: `libezflixdecoder.a`.

7.3.1 Data Stream run-time library

This library (`libds.a`) contains the core modules for real-time streaming: reading from CD-ROM, demultiplexing, delivery to the “subscriber” threads, and synchronization.

7.3.1.1 READ THIS NOW!!!

- API change: Removed `DSGetClock()`, `DSSetClock()`, `DSClockSync()`, `DSIsMarker()`, `DSSetBranchDest()`, and `DSSetBranchType()`. The first two were replaced by branch-aware procedures `DSGetPresentationClock()` and `DSSetPresentationClock()`.
- You should use a new shutdown sequence to avoid a race condition: Disconnect the `DataAcq`, dispose the `DataAcq`, then dispose the `DataStreamer`. See `DismantlePlayer()` in any of the example applications.
- The client application should NEVER call `DSSetPresentationClock()`. (Applications used to call `DSGetClock()` to work around a `DataStreamer` bug.)

7.3.1.2 Notes

- The data stream thread now sets the presentation clock on flush-branch, `DSCConnect`, and stop/start with flush. (It also handles pausing and resuming the clock.) This means that most branching can be done even without an audio subscriber.

However, you still need an audio subscriber to set the clock on non-flush branches. This requires continuous audio data up to the branch start point and immediately after the branch end point.

- API change: Added `DSClockLT()` and `DSClockLE()` routines, which compare two presentation clock {branchNumber, time} pairs.
- The `ControlSubscriber` and `ControlMaker` tool are gone. In their place, the Data Stream thread responds to `STRM STOP` and `STRM GOTO` chunks (without the race conditions inherent in the `ControlSubscriber`).

The Weaver script commands “`writestopchunk`” and “`writegotochunk`” will generate these chunks. A `STOP` chunk stops (pauses) when the subscribers finish playing back the data up to that chunk. A `GOTO` chunk branches the stream, playing the data up to that chunk if it's a non-flush branch.

- API change: `DSGetChannel()` and `DSSetChannel()` now use an unsigned status field. All subscribers support the `DSSetChannel()` mask argument so you can set and clear status bits.

- The streamer can now branch to any chunk in a stream; not just to the start of a stream block. This can save stream filler space and bandwidth but it has a cost. Branching into the middle of a stream block reduces seek coverage (increases effective seek time) because the first post-seek block is read, and then the first part is discarded.
- The header file "preparestream.h" now defines the same enable-audio-channel mask as the Weaver uses as a default. This constant is used by application code (such as the example apps) that tries to cope with a missing stream header chunk. In practice, you should always use the Weaver's "writestreamheader" command, to write a stream header chunk to every stream file.
- The trace logging code and compile-time switches were improved.
- API change: the streamer, dsblocksize, and marker tables now use unsigned file sizes and offsets in order to span up to about 1 DVD disc layer.
The Weaver, however, has not been updated to generated uint32 marker table values.

7.3.1.3 Known Bugs and Caveats

- The trace logging facility (for real-time debugging) is set to print its logs to the debugger's Terminal window.
To write the logs to files instead, set the compile-time switch `USE_RAW_FILE` in `subscribertraceutils.c` to "1".
- `DSGoMarker(..., GOMARKER.BACKWARDS)` will return a range error if the stream is beyond the last marker point. This bug will be fixed in the 3.0 release.

7.3.2 Subscriber library

This library (`libsubscriber.a`) is a collection of individual "subscribers," each of which specializes in handling a specific type of data. Each subscriber started by an application exists as a separate thread and receives data in time-stamped "chunks" from the Data Stream thread after demultiplexing. This release includes five subscribers:

- SAudio
- MPEGVideo
- MPEGAudio
- EZFlix
- Data

You may also create your own specialized subscribers by using the provided source files as a model.

7.3.2.1 READ THIS NOW!!!

- This release includes two new subscribers: the MPEGAudio Subscriber and the DATA Subscriber.
- The Join Subscriber, which prior to M2 was used for streaming background data, is no longer included. It has been replaced by the DATA Subscriber, which is more capable.
- The ControlSubscriber has also been deleted. Its functionality is replaced by new functionality in the DataStream thread.

7.3.2.2 SAudio Subscriber

Enables streamed playback of native 3DO sampled audio formats, i.e., any audio format that can be handled by the AudioChunkifier tool (see Data Streaming tools, below). Passes audio chunks to the Audio Folio for decompression, any other processing, and then for routing to the 3DO M2 audio output jacks.

Known Bugs and Caveats

- This subscriber supports multiple channels but, because of the limitation of 23 signals per thread, do not use more than four channels simultaneously.
- The sound spooler allocates a signal for each sound buffer allocated. When multiple channels are active at the same time, multiple sound spoolers are created which may cause the 23 signals per thread limitation to be exceeded. To avoid this, reduce the number of audio buffers allocated per sound spooler by
 - Specifying the number of audio buffers as four (the default is eight) when chunkifying your audio data (i.e., AudioChunkifier -cs 4 -i sample_audio.aiff -o sample_audio.audio) or by
 - Changing the number of buffers using the SAudioTool on a chunkified audio file (i.e. SAudioTool -nb 4 -i sample_audio.audio).

7.3.2.3 MPEG Video Subscriber

Decodes MPEG-1 video streams, manages the MPEG device interface, and hands decoded frames to the application for display.

Notes

- API change: <streaming/fmvdriverinterface.h> has been privatized by moving it to
libs/streaming/Subscribers/MPEGVideoSubscriber/.
The audio portions have been removed because they no longer apply now that the MPEG audio decoder is a DLL.
- API change: The struct "VideoSequenceHeader" in <misc/mpeg.h> now includes the quantizer matrices.
- The MPEGVideoSubscriber now passes the MPEG PTS (presentation time stamp, at 90 kHz) to the client in the buffer->pts.FMVOpt_PTS field of the filled-frame-buffer message. This is done for test programs and for apps, such as VideoCD, that need to display the elapsed time.

7.3.2.4 MPEG Audio Subscriber

The MPEG Audio Subscriber decodes MPEG audio elementary streams, manages the interface to the SW decoder, and enables streamed playback of the decompressed MPEG audio.

READ THIS NOW!!!

An MPEG audio chunk holds one compressed audio frame, so it typically consists of only 1274 bytes including chunk header. The streamer needs a subscriber message for each chunk that is outstanding at a subscriber, plus a couple more per subscriber for other requests. Otherwise, the streamer will run out of subscriber messages and abort stream playback.

Consequently, to play a stream file containing only MPEG audio chunks, either decrease the total stream buffer space (e.g. 20K bytes/buffer * 4 buffers should be plenty), or increase the number of subscriber messages to a few more than the number of chunks that will fit in the total stream buffer space at the same time.

7.3.2.5 EZFlix Subscriber

This subscriber decodes EZFlix video streams, manages the interface to the SW decoder, and puts decompressed frames into a display buffer for the application to display. EZFlix is a 3DO-proprietary SW video codec.

An important feature of EZFlix is its very small memory footprint -- only 3 KBytes, not including the output frame buffer.

EZFlix uses all key frames, so its data rate (bits/pixel) is higher than MPEG's.

Quality is good compared to other SW video codecs, but is usually lower than MPEG quality. Artifacts have a grainy look, which can be desirable artistically. (View the BladeForce example stream included with the EZFlixPlayer example app.)

READ THIS NOW!!!

- Programs that use the EZFlixSubscriber must link in the EZFlix Decoder library: `libezflixdecoder.a`.

Notes

- Removed `SendFreeEZFlixSignal()` and the channel # arg to `FlushEZFlixChannel()` and `DestroyEZFlixStreamState()`.
- Privatized some type and constant definitions and deleted others.

Known Bugs and Caveats

- This subscriber uses an older method of processing, in which it does not begin decompressing a frame until it is almost time to display it. This can cause a delay before the image actually appears on the screen.

7.3.2.6 DATA Subscriber

Enables the application programmer to put arbitrary data into a data stream. The DATA subscriber copies data from the stream buffers into application memory, optionally combining any number of smaller chunks into a larger chunk, and optionally decompressing the data as it is copied.

A block of user data may be divided into any number of DATA subscriber chunks, which will then be reassembled at runtime before being delivered to the application. This allows the programmer to stream data which would otherwise be too large (i.e. a piece of data which is larger than a stream block), and to minimize FILL chunks by creating small DATA subscriber chunks (see DATAChunkify tool documentation).

The amount of stream space occupied by DATA chunks may be further reduced by compressing the data before it is chunked (see comp3DO documentation). When it encounters compressed data, the DATA subscriber calls the compression folio to automatically decompress the data directly into the application's buffer as it is copied from the stream buffer.

7.3.3 DS Utils library

This library (`libdsutils.a`) contains four utility modules. These modules are needed by the other Data Stream libraries and they may also be useful to other programs and used as example source code.

7.3.3.1 *dsblockfile.c*

A slightly higher-level API to block-oriented file I/O.

Notes

- API change: `dsblockfile` now uses unsigned file sizes and offsets in order to span up to about 1 DVD disc layer.

7.3.3.2 *mempool.c*

A specialized memory allocator. A mem-pool holds a pool of uniform-sized memory nodes that can be used and reused very rapidly in a real-time program, like the streamer, without risk of memory fragmentation. The nodes can be pre-initialized to save work during each reuse. Pre-initialization often includes allocating resources such as an `IORequest` Item for each pool node.

7.3.3.3 *msgutils.c*

Minor utilities for dealing with OS messages and message ports.

7.3.3.4 *threadhelper.c*

A `NewThread()` routine for forking off a thread with some non-default tag args.

7.4 Data Streaming Examples

These consist of four example applications that illustrate how to use the Data Streaming run-time APIs and demonstrate some of M2's capabilities, such as MPEG.

The installer will put these apps onto your Mac hard drive in the remote folder (`...Remote:Examples:Streaming:`). One or more sample playable streams are provided for each example.

7.4.1 Notes

- To enable source-level debugging, change the makefile variable `"DebugFlag"` from `"0"` to `"1"`.
- The example applications now use the new shutdown sequence in `DismantlePlayer()`, which avoids a race condition.

The sequence is as follows: disconnect the `DataAcq`, dispose the `DataAcq`, and then dispose the `DataStreamer`.

You should use this sequence in your applications.

7.4.2 Fixed Bugs

- Some bugs were fixed in these example applications; for example, the error-handling code is now careful to set the variable `"status"` before calling `FAIL_NEG()` because the failure case jumps to the label `"FAILED:"`, which leads to code that relies on the `"status"` variable.
- The example applications no longer use the obsolete `Control` subscriber.
- Some Portfolio link libraries became DLL modules, so the `"make"` files for the example applications were updated.

7.4.3 PlaySA

Uses the SAudio Subscriber and the MPEG Audio Subscriber to play native 3DO sampled audio and 3DO MPEG audio stream files, which can contain up to 4 audio channels multiplexed together. During playback, the control pad can be used to switch between any of the channels.

7.4.3.1 Notes

- This example application can now play a stream that contains either one of SAudio data or MPEG audio data.

7.4.3.2 Known Bugs and Caveats

- For MPEG audio stream files, PlaySA can only play all channels simultaneously.

7.4.3.3 Fixed Bugs

- Pressing the C button during PlaySA will now rewind the stream to the beginning, even if you are at the end of the stream.
Formerly, if you pressed the C button during PlaySA near the end of the stream, after all the data was delivered but not yet played out, the program would quit instead of rewind.

7.4.4 VideoPlayer

The VideoPlayer example application uses the MPEG Video Subscriber and MPEG Audio Subscriber or SAudio Subscriber to play a stream file containing multiplexed MPEG video and MPEG or native 3DO audio. It plays back audio and video with solid sync.

7.4.4.1 Notes

- This example application demonstrates both flush- and non-flush branching. With 32K or smaller stream buffers, streaming from hard disk, and a stream that contains audio at the very beginning, you can rapidly click the control pad's left-arrow to demonstrate rapid branching. Click shift-left-arrow to demonstrate a non-flush branch. You will see a delay in the visible response, where the delay depends on the total stream buffer space.
If you do a non-flush-branch, the Data Streamer cannot set the clock. Consequently, the MPEG Video Subscriber will play slowly until the audio subscriber sets the clock. If the stream has no audio data, you can press left-arrow to do a flush-branch so the streamer will set the clock and playback will return to normal speed.
- VideoPlayer now accepts the command-line option "-l" or "-loop" to make it loop.
- If the stream playback stops due to a STOP chunk, VideoPlayer will print a message in the debugger's Terminal window. VideoPlayer will resume playback if you press the Play/Pause (>/| |) button.

7.4.4.2 Known Bugs and Caveats

- This app does not turn on horizontal or vertical interpolation.
This app ought to use the BMTAG_BUMPDIMS tag and the bumped bitmap dimensions instead of the hard-coded dimensions.
- If the image is smaller than 320x240, it places it in the upper left-hand corner of the display rather than centering it.

7.4.5 EZFlixPlayer

Uses the EZFlix Subscriber and the SAudio Subscriber to play a stream file containing multiplexed EZFlix video and native 3DO audio. Plays back audio and video with solid sync. Centers the frame in a 320x240 frame buffer, which is then expanded by the Graphics Folio to 640x480 size for display.

7.4.5.1 Notes

- You can test flush-branch to start by pressing left-arrow, and non-flush branch to start by pressing up-arrow. (Left-arrow no longer un-pauses when it branches to start.)

If you do a non-flush-branch, the Data Streamer cannot set the clock. Consequently, the EZFlixSubscriber will wait for the audio subscriber to set the clock.

If the stream has no audio data, it will just wait. You can press left-arrow to do a flush-branch, which will cause the streamer to set the clock and playback to resume.

7.4.5.2 Known Bugs and Caveats

- When using MemDebug, you may get the following error message when using EZFlixPlayer:

Data Access Fault

This is caused by a memory access bug in the clean-up code of `EZFlixPlayer.c`. This bug will be fixed in the next release.

7.5 Data Streaming Tools

These are a set of MPW tools for converting standard audio and video data formats into a real-time, M2-playable stream. Audio and video files begin as some standard format, either compressed (e.g., MPEG Video) or uncompressed (e.g., QuickTime "raw" video or AIFF audio).

If uncompressed, the file is first compressed, if necessary, using either a tool provided by 3DO (e.g., real-time MPEG encoding station, Sparkle or SquashSnd) or by another vendor or service (e.g., MPEG encoding).

Next, the file is "chunkified" (i.e., packetized) by one of the tools described below, in preparation for "weaving" (i.e., multiplexing).

Finally, the chunk files are input using a "weave script" to the Weaver, a Data Streaming tool which multiplexes the chunk files into a playable stream.

Each of these tools will provide a usage statement if invoked in MPW with no arguments.

7.5.1 READ THIS NOW!!!

- Some names have changed.
 - MovieToEZFlix to EZFlixChunkifier
 - MovieToStream to QTVideoChunkifier
 - SFToStream to AudioChunkifier
- MPEGChunkifier was replaced by MPEGVideoChunkifier and MPEGAudioChunkifier.
- Chunkify, the Join Subscriber's tool - no longer exists. Use the DATA Subscriber instead.

- All stream tools were built using E.T.O #19, MPW version 3.4.1b4.

7.5.2 AudioChunkifier

Formerly known as SFToStream, this MPW tool converts a standard AIFF sampled-audio file, or an AIFC file (an AIFF file compressed using the 3DO audio tool SquashSnd), into a chunk file.

7.5.2.1 READ THIS NOW!!!

- Though the AIFC formats provide less compression than MPEG Audio, their run-time decompression consumes virtually no PowerPC CPU cycles, because decompression takes place in the 3DO M2 DSP (or in other 3DO M2 HW in the case of SQS2). MPEG Audio decoding, however, will consume a substantial portion -- 25% or more -- of the CPU.

7.5.2.2 Known Bugs and Caveats

- Currently supports input formats with the following choices of parameters, although the 3DO M2 Audio Folio does not support all possible parameter combinations (see "format restrictions" below). Also, testing priority has gone to 16-bit formats, which we expect to be more widely used than 8-bit formats for reasons of fidelity. See SquashSnd documentation and its usage message for more information on the AIFC compression options.
 - 44100 or 22050 Hz sample rates;
 - stereo or mono;
 - 8 or 16 bit per sample quantization;
 - uncompressed (AIFF) or one of 3 compressed (AIFC) formats:
 - SQS2: 2:1 compression; decompression with M2 HW;
 - CBD2: 2:1 compression; decompression with M2 DSP SW;
 - ADP4: 4:1 compression; decompression with M2 DSP SW.
- Format restrictions:
 - SQS2 compression applies to mono only. (CBD2 works for both mono and stereo.)
 - ADP4 compression applies to mono only.
- SQS2 vs. CBD2 for mono: since SQS2 decoding is HW-based, it takes almost no DSP resources (ticks or program space), but CBD2 may produce slightly better audio fidelity, and CBD2 can be used for stereo.
- ADP4 currently may not be an optimal ADPCM implementation, due to the level of audio distortion perceptible in some material.

7.5.3 SAudioTool

An MPW tool which prints out the header information of a chunkified audio file. It can be used to modify audio chunk-file header information such as the logical channel number, initial amplitude, initial pan, and the number of chunks that the Audio Subscriber queue up to the Audio Folio at a time.

7.5.4 MPEGVideoChunkifier

The MPEGVideoChunkifier (previously called the MPEGChunkifier) is an MPW tool that converts an industry-standard MPEG-1 Video Elementary Stream into a chunkified version of same. Provides both prompt mode (-prompt option) and command-line mode MPW interfaces. See usage, help, and examples by invoking 'mpegvideochunkifier' in MPW with no arguments.

7.5.4.1 READ THIS NOW!!!

- To support branching in MPEG, files generated by MPEGVideoChunkifier have a different format than files generated by MPEGChunkifier. The new file format and the new version of the Data Streaming library are not backward compatible. **You must rechunkify your MPEG video files to use them with this release of the MPEG Video Subscriber.**

7.5.4.2 Notes

- MPEGVideoChunkifier has a different set of command line options. See the reference manual for more info.
- Branching tips.

The -p (progress reporting) option in MPEGVideoChunkifier causes information to be printed out that can be used to determine branch destinations.

For Example:

If you specify

```
MPEGVideoChunkifier -p -i disk:movies:file.mpg -o  
disk:movies:file.mpg -d 24 -s 24
```

Then you will get the following information:

Video frame rate = 23.976000

Frame Number	Frame Type	Frame Size	Decode Time	Presentation Time

83:	B	4888	820	844
84:	P	8792	830	884
85:	B	4824	840	864
86:	B	5016	850	874
87:	I	21848	860	914
88:	B	4632	870	894
89:	B	5272	880	904
90:	P	9176	890	944

From this information, you would conclude that Frame 87 (I Frame) is a good candidate for a branch destination.

Consequently, 860 (Decode Time) can be used in your weave script as the argument for "markertime".

7.5.4.3 Known Bugs and Caveats

- If you omit the -fps (frames per second) command-line option, MPEGVideoChunkifier will assume the output frame rate should be that which is specified in the video sequence header of the input MPEG-1 Video Elementary Stream. This is an enumerated parameter, for which only the following frame rates are defined by the MPEG standard: 23.976, 24, 25, 29.97, 30, 50, 59.94, or 60 fps. Use the -fps option to force MPEGVideoChunkifier to make the output frame rate a different value. It will neither add nor subtract frames from the input file, it will only make their frame rate different. This feature is necessary, for example, if you want to use Sparkle to encode QT movies with non-MPEG-standard frame rates (e.g., 12, 15, 20 fps), and if you want to perform the encoding in such a way as to preserve the non-standard frame rate. In this case, you must know the non-standard frame rate of the MPEG Video stream, and specify it to the MPEGVideoChunkifier explicitly, using the -fps option.
- MPEGVideoChunkifier is a fairly easy-to-use tool for weaving simple, linear, stand-alone MPEG movie clips. For such simple streams, MPEGVideoChunkifier provides adequate defaults for most of its input parameters, and you need not learn about the details. But for preparing more complex streams, e.g., preparing separate clips which will later be concatenated, or preparing branch points, it is important to understand the command-line parameters in detail.

7.5.5 MPEGAudioChunkifier

The MPEGAudioChunkifier is an MPW tool that converts an industry-standard MPEG-1 Audio Elementary Stream into a chunkified version of the same. This tool provides both prompt mode and command-line mode MPW interfaces. See usage by invoking "mpegaudiochunkifier -help" in MPW.

7.5.5.1 Known Bugs and Caveats

- Known bug: Invoking mpegaudiochunkifier with -help or incorrect arguments will print usage, which may be missing the command name, or may print garbage where the command name should be printed.
- See "MPEG Audio Subscriber" on page 35 regarding stream block size and subscriber messages.

7.5.6 EZFlixChunkifier

Formerly known as MovieToEZFlix, this is an MPW tool which takes as input a "raw" QuickTime movie file and both compresses it using 3DO's proprietary EZFlix encoder, and chunkifies it so that the output file is ready to weave.

7.5.6.1 READ THIS NOW!!!

- Using the -h and -w options to crop a movie can cause a crash. This bug will be fixed in the next release.
Workaround: This bug does not happen if all frame dimensions are multiples of 16.

7.5.6.2 Known Bugs and Caveats

- You must specify the desired frame size using the -h and -w options or else frame size 256x192 may be assumed.

Note that all frame dimensions must be multiples of 16 because of the aforementioned bug.

- The -k option, to flag key frames, has been removed.
- The -q option allows you to set the quality level. Use 25% for an Opera level of quality. Use 75% if you can accept the higher data rate.
- On PowerPCs, garbage characters may be seen in the verbose output when altering the height and width making the output file unusable.

7.5.7 QTVideoChunkifier

Formerly known as MovieToStream, this is an MPW tool which chunkifies a QuickTime movie that has been compressed using the EZFlix QT component.

The process of (a) starting with a raw QT movie, (b) compressing it with the EZFlix QT component using some QT-based tool, and (c) converting it to a chunk file with QTVideoChunkifier is equivalent to the process of (a) starting with a raw QT movie and (b) using EZFlixChunkifier to simultaneously compress and chunkify it.

7.5.7.1 Notes

- QTVideoChunkifier only accepts an EZFlix-compressed QuickTime movie file as input. Since EZFlix is the only SW codec currently supported by 3DO M2 Data Streaming, it's the only type of QT movie file you should be using as input.

7.5.7.2 Known Bugs and Caveats

- Known bug: The QTVideoChunkifier experiences problems on the PowerMac due to an incompatibility between the EZFlix QT Component 1.0b1 and the QuickTime™ 2.0 component. To resolve this problem, either use a utility like "Thing Motel" to install the EZFlix Component, or use a more recent version of QuickTime™ (2.1 or later).
- The -k option, to flag key frames, has been removed.

7.5.8 DATAChunkify -- the DATA Subscriber's Data Prep Tool

DATAChunkify is an MPW tool that converts arbitrary data files into DATA subscriber chunks. For details about the tool, read the man page, or invoke the tool with "DATAChunkify -help" or "DATAChunkify -?" in MPW.

Known Bugs and Caveats

- This tool DOES NOT do data compression. If the "-comp" option is used, the data is assumed to have been compressed with the comp3DO MPW tool, and the chunk header is set so that the DATA subscriber will decompress the data when it is read from the stream.

7.5.9 Weaver

An MPW tool which multiplexes (“weaves”) one or more input chunk files into a single M2-playable stream file. Requires a “weave script” to specify the chunk files to be woven, as well as any control information required for the output stream file. See the chapter entitled “Weaver Script Commands” in the *3DO M2 DataStreamer Programmer’s Reference*.

7.5.9.1 Notes

- To find good branch points in an MPEG video stream, use MPEGVideoChunkifier’s “-p” (verbose progress report) option. This will display each frame’s frame type (I, P, or B), DTS (delivery timestamp), and PTS (presentation timestamp).

Pick an I-frame near the desired PTS, and then add a “markertime” command for that frame’s DTS. You must set the marker point at the frame’s DTS to cause that compressed frame to be delivered after a branch to that marker. An alternative is to compress each stream segment separately so that each stream segment begins with a closed GOP.

7.5.9.2 Known Bugs and Caveats

- Weaving a stream containing lots of small chunks (e.g., 1 KByte or smaller) with a stream containing fewer large chunks (e.g., half the streamblocksize or larger) can result in a stream where the large chunks fall increasingly behind the small chunks, according to their timestamp order.

This bug no longer occurs when the larger chunk type can be split across streamblock boundaries. See “Fixed Bugs” on page 45.

However, when large chunks cannot be split (e.g. EZFlix video), smaller chunks of other streams (e.g. audio) can still move ahead of them.

If enough small chunks move far enough ahead to make playback unsmooth or blocked up (you can verify this with DumpStream), increase the audio chunk size (e.g. to 4 KBytes or more).

This problem might be severe if EZFlix video were used with MPEG audio, but that combination is not recommended because both those decoders require big percentages of the CPU cycles.

- To set the starting presentation time of an MPEG video elementary stream, use the MPEGVideoChunkifier’s “-s” option, NOT the relative starting time in the Weaver-script “file” command.

Always set the relative starting time in the Weaver-script “file” command to “0” for an MPEG video chunk file. The “file” command’s relative starting time argument can only adjust the delivery timestamps (DTSs), not the MPEG video presentation timestamps (PTSs). Attempting to use it to adjust MPEG PTSs would make playback unsmooth and unsynchronized.

- Weaver scripts should ALWAYS include the “writestreamheader” command. Many of the parameters specified by the Weaver script cannot be rendered effective unless the Weaver creates a stream header containing them. It will only do so if this command is included in the script.
- The Weaver’s “writemarkertable” command adds a spurious first marker, which points to a chunk in the first stream block, e.g. the third header chunk. (Unlike all other markers, this one is not followed by a FILL chunk out to the end of the stream block.) Having an extra marker affects some of the go-marker run-time operations.

-
- The “writegotochunk” command only works when its options argument is “1”. It won't accept additional option flags or alternative branch types.

7.5.9.3 Fixed Bugs

- Fixed a bug in the Weaver's chunk-ordering logic, in which weaving a stream containing lots of small chunks (e.g., 1 KByte or smaller) with a stream containing fewer large chunks (e.g., half the streamblocksize or larger) can result in a stream where the large chunks fall increasingly behind the small chunks, according to their timestamp order.

The fix works when the larger chunk type can be split across streamblock boundaries. Since the only stream chunk type which can be split is MPEG-Video, this fix works only for MPEG-Video.

However, weaving MPEG-Video (which typically has larger chunks) with MPEG-Audio (which typically has small chunks of 400-500 bytes) is the most likely situation in which this problem would occur. The problem no longer happens for this situation.

7.5.10 Dumpstream

An MPW tool which will dump out information about the contents of a woven file. Depending upon which options are specified, output information may include an ordered list of all chunks contained in the stream, with the chunk type, subtype, timestamp, size, and byte offset of each chunk.

7.5.11 Worksheet and Exercises

On the 3DO M2 2.0 CD, you will find source code for the four Data Streaming example applications, as well as example scripts and data for chunkifying and weaving playable streams. See the folder with pathname:

Examples:M2_2.0:Streaming:

Within this folder, in addition to the four source-code folders, you will find a text file named “ds_examples_readme” and a folder named “Tools_and_Data”.

The readme file explains the example apps and streams contained in the overall folder.

The Tools_and_Data folder contains an MPW worksheet named “Video.worksheet”, which shows how to use the various chunkifying and weaving tools to create playable streams. Tools_and_Data also contains various elementary streams (MPEG-Video, MPEG-Audio, AIFF audio, and EZFlix video) for use with the worksheet examples.

7.5.11.1 READ THIS NOW!!!

- The installer will NOT install this folder on your Mac, because of the large size of some of the files.

7.6 Video Tools

7.6.1 3-2 Pulldown, MovieEdit, MovieCompress

7.6.1.1 Notes

- MovieCompress updated since 1.1, version 1.3b1
- Defaults to EZFlix compression when available.
- Error handling is now more robust. Error dialogs report the error number.

- A default file name is provided by appending the four-character code of the codec.

7.6.1.2 Known Bugs and Caveats

- **MovieEdit:** If you attempt to save a movie from the same directory that has the same name as a movie that is also open, both movies will end up invalid. The work around is to close the original movie before replacing it.

7.6.2 EZFlix QT Component 1.0b1

7.6.2.1 Notes

- Quality settings have been relabeled as follows:

0 -> 10, 25 = 25, 50 -> 4, 70 -> 60, 100 -> 75.

7.6.3 Sparkle 2.4.5

Sparkle is an MPEG encoder/decoder for the Mac. It requires SoundLib on PowerMacs, which is provided by SoundManager 3.1 which is shipping in this release as well.

7.6.3.1 Notes

- You can use Sparkle to create MPEG still images from PICT files. First, open and display the PICT file, and then Save As MPEG format.
- Updates since version 2.3.3 which is the version we've been shipping since the 1.0 software CD:
 - There's faster Power Mac playback now.
 - A lot of bug fixes, see the release notes posted with the tool on the CD for more detail.

7.6.4 SoundManager 3.1

Fulfills the need for SoundLib on Power Macs necessary for using Sparkle v.2.4.5.

8.0 Audio

8.1 READ THIS NOW!!!

- Some changes have been made that could theoretically break code but are unlikely to do so. They are:
 - SetItemPri() for audio instruments is now illegal because it messes up the DSP execution order.
 - CreateInstrument() now enforces the maximum value of 200 for AF_TAG_PRIORITY and will return an error if exceeded.
 - The following obsolete tags have finally been removed so any old code using these will need to convert to using the new tags. Note that the new tags take floating point arguments so you must use ConvertFP_TagData () to pass the argument.
 - AF_TAG_AMPLITUDE is gone, use AF_TAG_AMPLITUDE_FP
 - AF_TAG_SAMPLE_RATE is gone, use AF_TAG_SAMPLE_RATE_FP
 - AF_TAG_FREQUENCY is gone, use AF_TAG_FREQUENCY_FP
 - Note the following when using the Audio Patch Folio. Previously, the user used #include <:audio:patchcmd.h> to use the API from this folio,

but now the user must use `#include <:audio:patch.h>`. The user must also edit the make file to include the module `audiopatch`.

8.2 Notes

- Added support for 3D sound spatialisation.
- Added new family of query functions.
- Added new Beep Folio, which offers a low memory footprint, low functionality alternative to the Audio Folio.
- Implemented numerous optimizations and footprint reduction in the Patch system.
- Added pitch-based envelope time scaling.
- Now provide interactive examples of spooling music off disc with branching.

8.3 Known Bugs and Caveats

- Rev E boards will not work with Release 2.0.
- Envelope time scaling does not affect envelope loop times set by `AF_TAG_SUSTAINTIME_FP` and `AF_TAG_RELEASETIME_FP`.
- `SampleInfoToTags()` fails to transfer `smpl_Detune` to an `AF_TAG_DETUNE` tag. This also affects `LoadSample()`. The result is that any detune information stored in an AIFF is not copied to the resulting Sample Item.

8.4 Fixed Bugs

- "sampler_drift_v1.dsp" now resets Drift output to zero when restarted.
- Reset output of "schmidt_trigger.dsp" when restarted.
- Fixed tick allocation of "filter_101z.dsp", updated man pages.
- `CreateMixerTemplate()` - Fixed DSP code for mixers with only 1 Input.
- A knob created using the `AF_TAG_TYPE` option clipped its values based on the default type instead of the new type.
- `AF_TAG_TIME_SCALE_FP` now works properly in `StartInstrument()` and `ReleaseInstrument()`.

8.5 Beep Folio (new)

The Beep Folio provides a low memory footprint alternative to the Audio Folio. It uses a fixed synthesis architecture instead of a flexible dynamic patching system like the Audio Folio. See manual and man pages for more information.

Examples in `examples/audio/beep`.

8.6 Audio folio

8.6.1 Notes

8.6.1.1 Envelope Time Scaling

Added pitch-based time scaling for envelopes. This allows you to specify that an envelope plays faster and shorter as one goes up in pitch. This models acoustic instruments like pianos where the high notes are shorter than the low notes.

Tags for envelope:

`AF_TAG_BASENOTE` - Note at which pitch based time scale value = 1.0.

AF_TAG_NOTESPEROCTAVE - Number of semitones at which pitch time scale doubles.

AF_ENVF_LOCKTIMESCALE. When set, causes the Time Scale for this envelope to be locked at 1.0. Then using AF_TAG_TIME_SCALE_FP in StartInstrument() will have no effect on this envelope. This is useful if you are using multiple envelopes in an instrument and want some to be time scaled, and some not to be. Envelopes used as complex LFOs are often not time scaled. Note that this does not affect pitch-based time scaling.

8.6.1.2 Added Instrument query function family

GetInstrumentResourceInfo() - get instrument resource usage information.

DumpInstrumentResourceInfo() - display instrument resource usage to debug console.

GetNumInstrumentPorts(), GetInstrumentPortInfoByName(), and GetInstrumentPortInfoByIndex() - get instrument port information. These functions replace GetNumKnobs() and GetKnobName().

GetAttachments(), GetNumAttachments() - get information about attachments made within a patch.

8.6.1.3 Added other miscellaneous query support

GetAudioSignalInfo() - get information about audio signal types.

GetAudioResourceInfo() - get information about available DSP resources.

Added Attachment and Envelope support to GetAudioItemInfo().

8.6.1.4 Velocity Zones for MultiSamples

StartInstrument() now honors the LowVelocity and HighVelocity values for a Sample item. It will now use the first Sample attached to the Instrument whose range includes the specified velocity. This allows developers to use "velocity maps" with multisamples.

Known Bugs and Caveats

- There is a known bug with velocity zones. In order for velocity zones to work properly, a pitch must be specified along with the velocity when calling StartInstrument(). Specify the pitch using AF_TAG_PITCH.

8.6.1.5 Velocity Response Curves

Added support for exponential velocity response curves for StartInstrument(). This provides a more natural feel to amplitudes for MIDI score playing. While it is possible to do these calculations outside the folio and then use AF_TAG_AMPLITUDE_FP, that would make it impossible to use velocity zones for multi-sample selection. New StartInstrument() tags:

AF_TAG_SQUARE_VELOCITY - Like AF_TAG_VELOCITY except Amplitude = (Velocity / 127.0)**2. This gives a more natural response curve than the linear version.

AF_TAG_EXPONENTIAL_VELOCITY - Like AF_TAG_VELOCITY except:

Amplitude = $2.0^{((\text{Velocity}-127)*\text{expVelocityScalar})}$.

The value expVelocityScalar is set using AF_TAG_EXP_VELOCITY_SCALAR.

AF_TAG_EXP_VELOCITY_SCALAR (float32) - Sets expVelocityScalar used by AF_TAG_EXPONENTIAL_VELOCITY. The default value is (1.0/20.0) which means that the Amplitude will double for every 20 units of velocity.

Modified score player to use AF_TAG_SQUARE_VELOCITY

8.6.1.6 Other Audio Folio Changes

- Trap priorities above 200 in CreateInstrument().
- GetAudioFrameCount() now returns a uint32 value instead of a uint16.
- Added ConvertAudioSignalToGeneric() and ConvertGenericToAudioSignal() to provide audio signal conversions.
- Moved patch compiler (CreatePatchTemplate()) out of the audio folio into its own folio: AudioPatch folio.
- LoadInsTemplate() now uses IFF folio.

8.7 DSP Instruments

8.7.1 Notes

- Instrument "envelope.dsp" now uses a shared subroutine to reduce code memory usage.
- The following new dsp instruments have been added:
 - add_accum.dsp
 - input_accum.dsp
 - multiply_accum.dsp
 - output_accum.dsp
 - subtract_accum.dsp
 - subtract_from_accum.dsp
 - times_256.dsp
 - delay4.dsp
 - filter_3d.dsp
 - chaos_1d.dsp
 - depopper.dsp - useful for reducing pops when switching sounds.

8.8 Audio shell commands

8.8.1 audioavail (new)

New shell command to display available DSP resources.

8.8.2 dspfaders

Dspfaders can chain multiple patches or instruments, plus other enhancements - see man pages.

8.8.3 insinfo (new)

New shell command to display instrument or patch information.

8.8.4 makepatch

8.8.4.1 Notes

- Added Coherence command (relates to PATCH_CMD_SET_COHERENCE).

- Added envelope time scaling support.
- Improved syntax checking.

8.9 AudioPatch folio (new)

8.9.1 Patch compiler (CreatePatchTemplate())

8.9.1.1 Notes

- Added PATCH_CMD_SET_COHERENCE to permit patch compilation optimization at cost of guaranteed signal phase coherence.
- Improved patch code generation. Patch DSP code is now smaller and more efficient in all cases.
- Corrected patch tick usage computation.
- Added trap for unsupported PatchCmds.
- Renamed <:audio:patchcmd.h> to <:audio:patch.h>.

8.9.2 PatchCmdBuilder

8.9.2.1 Notes

- Added cumulative error system to permit checking success of PatchCmd constructors just once. See GetPatchCmdBuilderError().

8.10 AudioPatchFile folio (new)

8.10.1 Notes

- Added EnterForm3PCH() and ExitForm3PCH() to give access to patch file parser to parsers for larger scope files (e.g., a score file).

8.11 Music library

8.11.1 Patches

8.11.1.1 Notes

- Moved patch file loader (LoadPatchTemplate()) into its own folio: AudioPatchFile folio. To use the API from this folio, the user must #include <:audio:patchfile.h> and also include the module audiopatchfile in the makefile.

8.11.2 3DSound

8.11.2.1 Notes

- This release of the music library includes support for sound spatialization and localization. You can take an arbitrary mono sound source (a sample, soundfile, synthetic sound, patch, whatever), position it in 3-D space and move it around in real time, and the library calls will process the sound accordingly for you. You can select how much processing to do (and consequently how much of the available DSP resources to use) on a sound-by-sound basis. You can "animate" as many sounds this way as you have resources for.
- There are four examples using 3DSound included in the Examples:Audio directory: ta_steel3d, ta_bee3d, ta_sound3d, and ta_leslie3d.
- For more information, refer to the manual section on 3DSound.

8.11.3 Sample loader

(LoadSample(), GetAIFFSampleInfo(), <audio:parse_aiff.h>).

8.11.3.1 Notes

- Now uses IFF folio.
- Revised SampleInfo structure.
- Move prototypes for SampleItemToInsName() to and SampleFormatToInsName() prototypes from <:audio:score.h> to <:audio:parse_aiff.h>
- Added LoadSystemSample() to avoid using \$audio for loading sinewave.aiff.

8.11.4 Sound player

8.11.4.1 Notes

Now uses IFF folio for AIFF files.

8.11.5 Sound spooler

8.11.5.1 Known Bugs and Caveats

- Worked around the sample length alignment checks in the audio folio, which could cause perfectly good buffers to be rejected. However, this workaround can permit illegal buffer lengths to be played by the audio folio. A more reliable system to prevent buffer length alignment problems is being investigated.

8.12 Audio Examples

8.12.1 Notes

- Revived dormant Juggler examples: tj_simple.c, tj_multi.c, and tj_canon.c.
- New example playsf.c demonstrates how to loop a soundfile off disc. The spooling is handled by a separate task that is controlled via message passing.
- MarkovMusic example shows how to use Markov Chains to do algorithmic branching between markers in multiple sound files.

This example program assumes that there are four aiff soundfiles in the same directory with the names "spA.aiff", "spB.aiff", "spC.aiff" and "spD.aiff".

The files can be found in the release directory "Examples:Audio:MarkovMusic:Data". These files are not installed as part of the standard installation because they are so large.

To run the example, first copy these aiff files into your "Examples:Audio:MarkovMusic" directory.

Appendix A Microcard

The microcard is used to save game states.

A.1 MicroCard Usage

Since microcards have a limited life expectancy of no more than 10,000 writes and are shared devices among all titles, they must be treated as a limited resource.

Applications must avoid performing too many unnecessary operations that cause writes, such as application data writes, making directories, making files, removing files, removing directories, etc. The limit of 10,000 writes is specified by the hardware manufacturer. Results of write operations over that limit are unpredictable. This means that applications can update their private data files safely only 10,000 times, assuming that they do not increase the size of the file.

Note that some operations on Acrobat filesystem cause writes to specific system blocks of the microcard. These operations update the filesystem metadata, which is subject to the same limitation as user data.

Listed below are relevant Acrobat filesystem operations and their corresponding cost in term of the number of writes generated on behalf of the caller. "SYSTEM WRITES" is the total number of writes required by the filesystem to complete the transaction. "LIFE COST" is the number of writes (or fraction thereof) on the most used system block. All numbers exclude the cost of transaction processing and are approximations in an average working environment.

OPERATION	SYSTEM WRITES	LIFE COST
-----	-----	-----
CreateFile,		
CreateFileInDir	9-13	0.5-1.5
CreateDirectory,		
CreateDirectoryInDir	11-15	1.0-2.0
DeleteFile,		
DeleteFileInDir	3-6	0.5
DeleteDirectory,		
DeleteDirectoryInDir	5-8	0.5
FILECMD_ALLOCBLOCKS	2-4	0.5
FILECMD_SETEOF	1	0.5
FILECMD_SETTYPE	1	0.5
FILECMD_SETBLOCKSIZE	1	0.5

A.2 How to Prolong the life of the MicroCard

The Acrobat filesystem is the only filesystem that handles microcard transactions. Here are some guidelines to avoid overuse of microcards.

- Update game state infrequently. The fewer writes to the card, the longer the life of the card.

-
- When creating new files, immediately preallocate all the blocks that the file could ever require. It is better to use FILECMD_ALLOCBLOCKS only once for a given file, usually immediately after the file is created via CreateFile(). When files grow, the Acrobat filesystem creates a new transaction to protect the integrity of the system data in case the card is suddenly removed. These transactions are costly to the life of the microcard, since they are writes to various system blocks on the card. The lower the number of transactions, the longer the life of the card. Another advantage of preallocation is that it significantly reduces fragmentation.
 - Make directories and files once and limit the number of times they are renamed or deleted. Again, these kinds of operations generate new transactions, which translate to writes to the card.

Appendix B MPEG Read-This-First Roadmap

To have a productive and successful experience using MPEG in a 3DO M2 title, there are a number of things you must know about MPEG as a standard, about MPEG encoding and formatting, about 3DO M2 MPEG decoding support, and about a number of pieces of relevant software that are not all located in one place in this 3DO M2 2.0 release.

The purpose of this section is to introduce you to some essential aspects of MPEG, and to provide you a roadmap to the relevant MPEG-related software and documentation in this release.

IMPORTANT: If you don't have time to read this entire section of the release notes immediately, be sure to read at least section "**Interchange formats: MPEG-1 Video, MPEG-1 Audio, AIFF Audio**" on page 56 before you invest in any MPEG encoding services or tools.

B.1 MPEG standards

MPEG is the informal name of a set of ISO standards for the compression of digital video and audio data streams. These ISO standards explain the basic algorithms for MPEG compression and decompression processes, but what they really specify are the *encoded formats* for MPEG-compressed data streams.

This 3DO M2 2.0 release supports real-time decoding of standard MPEG-1 Video "Elementary" streams and Audio "Elementary" streams (i.e., streams containing only MPEG-1 Video or only MPEG-1 Audio), as specified by the video and audio parts of the MPEG-1 standard (ISO 11172-2 and ISO 11172-3 respectively).

Currently, we have no plans to provide an API for the direct playback of MPEG-1 "Systems" streams (ISO 11172-1). ISO 11172-1 is an ISO stream format for MPEG-1 Video and MPEG-1 Audio multiplexed together.

Note that MPEG-1 Video and MPEG-1 Audio are independent standards -- either can be used with or without the other.

B.2 MPEG Decoders

B.2.1 MPEG Video Device

The MPEG video device provides low-level MPEG 1 video decompression services using M2's built in MPEG hardware.

B.2.1.1 READ THIS NOW!!!

For video streamed from CD, applications should avoid using the MPEG video device directly and will find the DataStreaming MPEG library more convenient to use. Applications that wish to use MPEG for still image compression or that need more control over the decompression processes can call the MPEG device directly.

B.2.1.2 Notes

The following significant changes have been made to the MPEG video device in release 2.0:

- The device now supports decoding of multiple simultaneous streams. Each time the device is opened it creates and maintains a separate stream context and decoding thread, and the MPEG hardware is time-sliced between each

thread. This process is transparent to the calling thread(s) until the performance limit of the hardware is reached.

The current M2 hardware can decode approximately 200 320x240 pictures (or an equivalent number of 16x16 pixel macroblocks) per second peak (your actual highway mileage will probably be less).

- The device now provides improved support for decoding of still pictures, also known as I-pictures.

If the application sends a control command with the VID_CODEC_TAG_KEYFRAMES tag set after the device is opened and before any MPEG data has been written to the device, the device will not allocate any reference buffers. It will return decoded pictures immediately as they are decoded, instead of after the normal pipeline delay for reference pictures (I-pictures or P-pictures).

Only MPEG I-pictures can be decoded in this mode and any other picture types are ignored. This mode saves memory normally allocated by the device for reference pictures. Sample code for decoding still pictures can be found in the Photo example. See "MPEG Examples" on page 73 for notes on Photo.

- The device provides improved support for branching to a new stream or to a location within the same stream.
- Man Pages have been added for the MPEG video device commands.

B.2.1.3 Fixed Bugs

- Debugging `printfs` have been removed from the device driver.
- The device no longer returns an old or garbage picture when first opened or when transitioning to a new stream.

B.2.2 MPEG Audio Device

The MPEG Audio Device has been replaced with a DLL (see below).

B.2.3 MPEG Audio Decoder DLL

The MPEG Audio Decoder DLL replaces the MPEG Audio device. It provides MPEG 1 audio decompression services via a Dynamic Link Library (DLL) located in the folder "Remote\System.m2\Modules:mpegaudiodecoder".

The man pages for MPEG Audio functions appear in "Man Pages for MPEG Audio Functions" on page 60.

B.2.3.1 READ THIS NOW!!!

- This DLL decodes without hardware support, using only M2's PowerPC CPU. The algorithm uses approximately 35% of the CPU bandwidth and may negatively impact other CPU intensive operations such as 3D graphics. For this reason, it is recommended that graphics intensive portions of applications use one of the DSP-based audio compression methods instead of MPEG audio.

When the use of MPEG audio is appropriate, it is generally more convenient for the application to use the MPEG Audio Subscriber found in the DataStreaming subscriber library rather than calling the MPEG Audio Decoder DLL directly. See the DataStreamer documentation concerning the MPEG Audio Subscriber.

- The current version of the MPEG audio decoder supports layer II, 44.1 kHz, stereo, joint stereo, mono, and dual channel audio but does not perform error checking. Note that although the MPEG Audio decoder supports dual channel, the 2.0 version of the MPEG Audio Subscriber does not.

B.3 3DO M2 support for real-time MPEG-1 decoding

As described above, M2's support for MPEG-1 Video is provided by a real-time hardware decoder and its associated software interfaces, whereas M2's support for MPEG-1 Audio is provided by a software-only DLL implementation. Although there will be circumstances in which it is appropriate and desirable to use MPEG-1 Audio with MPEG-1 Video, this is not always the best combination, especially in environments that are CPU intensive.

B.3.1 Alternative to MPEG-1 Audio

As an alternative to MPEG-1 Audio, you might consider native 3DO sampled-audio formats, which provide 2:1 to 4:1 compression with excellent audio quality, and which consume virtually no PowerPC CPU cycles.

Developers may prefer native 3DO audio formats for many, perhaps even most, situations due to their low CPU usage. The real benefit of MPEG-1 Audio is for situations where the following conditions all hold:

- CD-quality audio is needed;
- CD bandwidth or capacity is at a premium;
- CPU cycles are plentiful.

For bandwidth comparison, a CD-quality stereo audio stream (44.1 kHz sampling rate) occupies about 31 KBytes/sec in MPEG-1 Audio format, vs. about 86 KBytes/sec in 3DO's 2:1-compressed audio format. Given that 2X CD bandwidth is 300 KBytes/sec, it is clear that MPEG-1 Audio is important for some but by no means all applications.

B.4 Encoding/formatting MPEG streams for 3DO M2 playback

The previous section gave a roadmap for topics on the playback of MPEG streams on M2, but not on creating the streams. This section provides a roadmap for the other half of the puzzle: encoding and formatting MPEG and audio data for playback.

B.4.1 Interchange formats: MPEG-1 Video, MPEG-1 Audio, AIFF Audio

Video and audio streams can begin life in any number of formats, but if your objective is to create an M2-playable stream containing MPEG-Video and synchronized audio, your first goal must be to convert your video and audio streams into these standard formats:

- MPEG-1 Video elementary stream, **square pixels recommended**.
- MPEG-1 Audio elementary stream, mono or stereo, sampling rate 44100 Hz.

- OR -

AIFF audio file, mono or stereo, sampling rate 44100 or 22050 Hz.

As discussed previously, whether you choose native 3DO (AIFF or AIFC) audio or MPEG-1 Audio depends on the needs and resources of your application.

If you decide to use MPEG Audio, then your task becomes to convert your audio

into an MPEG-1 Audio elementary stream. You can accomplish this by various methods, including real-time capture and MPEG Audio encoding using a commercial encoder or a service bureau, and offline MPEG Audio encoding from an compressed file of sampled audio. (In the latter case, there are various public-domain MPEG-audio encoders available on the Internet.) Once your data is in the MPEG Audio format, you will be ready to convert it into an M2-playable stream using the MPEGAudioChunkifier and Weaver tools.

If you decide to use native 3DO audio, your data-preparation task will probably be even simpler, as AIFF is a widely-used format for (uncompressed) sampled audio. 44100 Hz is the standard music-CD sampling rate, and 22050 hz is exactly half that rate. Once your audio data is in one of these formats, you can compress it using the 3DO SquashSnd tool (see *3DO M2 Tools for Sound Design*), and then convert it to a playable stream using the AudioChunkifier and Weaver tools. (See the "Data Streaming tools" section of these release notes, and the *3DO M2 DataStreamer Programmer's Reference*.)

Note: Although 3DO M2 can play back audio data sampled at rates other than 44100 and 22050 Hz, Data Streaming currently supports these two sample rates only.

For MPEG Video, once you have secured an MPEG encoder or service, and your video has been compressed into the MPEG-1 Video elementary stream format, you will be ready to convert it into an M2-playable stream using the MPEGVideoChunkifier and Weaver tools. (See the "Data Streaming tools" section of these release notes, and the *3DO DataStreamer Programmer's Reference*.)

Keep in mind, though, that the MPEG video story is not quite as simple as the audio case, for a number of reasons:

- The MPEG-1 Video algorithm is fairly complex.
- The MPEG-1 Video format ("syntax") has many internal options.
- MPEG encoders from different vendors may use aspects of the syntax in perfectly conformant, but different, ways.
- Shareware MPEG encoders do not undergo rigorous testing.

B.4.1.1 MPEG-1 Video format caveats

The "syntax" of the MPEG-1 Video stream format specified by ISO 11172-2 accommodates a number of parameters. While M2's MPEG-Video decoder has been designed to handle any legal combinations of these, some values can make your programming life more difficult, require extra computational resources, and produce a less satisfactory result than others. In particular:

- **Pixel aspect ratio**

MPEG-1 syntax allows many different pixel aspect ratios, but for 3DO M2 you should use **square pixels** (pixel aspect ratio = 1.0). For full-screen display, request 320x240-sized images, rather than the "SIF" size 352x240 used for the VideoCD MPEG-Video format.

If image data placed in the 3DO M2 display frame buffer does not have square pixels, the image will appear stretched on the TV screen. This is not to say that you **MUST** use square pixels. You may use non-square pixels, but if you do not want the image to appear stretched, you must re-sample the image using the texture-mapping capability of M2's graphics engine. However, you will get better quality and compression, and use fewer run-time resources, if you use a square

pixel format in the first place. VideoPlayer, the example MPEG movie player we have supplied in this release does not show how to do this resampling. (See the "Data Streaming examples" section of these release notes.)

- **Frame size**

Your life will be easier if you restrict both the width and height of the frames in your MPEG stream to **multiples of 16 pixels**. MPEG-1 syntax allows the video frame size to be just about anything, but internally images are divided into 16x16 sized "macroblocks." If dimensions are not mod-16, MPEG encoders must pad images accordingly, and MPEG decoders are supposed to trim off the padding prior to display. The 3DO M2 MPEG driver in this 3DO M2 2.0 release does not do trimming, nor does the "videoplayer" example app. Of course, if you use the full frame size 320x240, this problem does not arise.

- **Frame rate**

MPEG Video syntax is uncharacteristically restrictive about this parameter. It allows only the "video world" frame rates of 23.976, 24, 25, 29.97, 30, 50, 59.94, or 60 frames per second (fps). These may be just fine for most circumstances, but we believe other rates (15 fps, 20 fps, or really anything) should be allowed too. In this 3DO M2 2.0 release, if you want a "non-MPEG-standard" frame rate, you must manually set the frame rate to a different value, using the "-fps" MPEGVideoChunkifier command-line option (see release notes on this tool).

B.4.2 MPEG-1 Video encoding alternatives

There are a number of different MPEG-Video encoding options available. If you plan to use MPEG-Video in your 3DO M2 title, you should make this choice carefully, because the quality of the MPEG encoding system (the "cleanness" of video capture, the cleverness of the MPEG encoding algorithms, and other factors) can have an enormous influence in the quality of the resulting MPEG Video. Your range of options is as follows. (Some of these options are discussed in the *3DO M2 Video Processing Guide*.

- **3DO Real-Time MPEG Encoding System (MPEGXpress)**

You may purchase from 3DO a product called MPEGXpress – Mac-based real-time MPEG encoding system. The first version of this product is available now, and has won several awards for its high MPEG-Video quality, simple installation, and easy-to-use Mac interface. 3DO MPEGXpress supports (real-time) analog video input, as well as "off-line" encoding of software video formats such as QuickTime movies. 3DO MPEGXpress has been tailor-made to meet the needs of 3DO M2 developers, so you should consider it seriously.

- **Sparkle software MPEG encoder**

This is a shareware tool which 3DO has provided in this 3DO M2 2.0 software release. Sparkle takes a QuickTime movie file as input and outputs an MPEG-1 Video elementary stream. See the documentation in the *3DO M2 Video Processing Guide*, as well as the section on Sparkle elsewhere in these release notes, for complete information, but here are a few of the caveats:

- Sparkle has no rate control, so full frame-rate streams may not play smoothly.
- If P or B frames are used, Sparkle may use a feature of the MPEG-Video syntax called "macroblock_escape," which many MPEG encoders (including 3DO's MPEGXpress) will not utilize for images 512 or less pixels wide. The version of the 3DO M2 development-system hardware supported by this 3DO

M2 2.0 software release has a bug in its MPEG decoder, which will cause a stream containing a macroblock_escape to hang. This bug will be fixed in the Revision G development card. See the MPEG-Video device driver release notes.

- **MPEG Encoding services**

There are a number of service bureaus and post-production houses that will do MPEG encoding for a fee. Many of these do excellent work, and many are in the learning phase.

However, most of these services target their encoding for the VideoCD format, and so may not offer features which are best for M2. In particular, many can only encode video to the 352x240 non-square-pixel format, which will require you to resample the images before display. Also, most of them have real-time encoders which can only handle analog-input video. If your source is rendered video in softcopy format, the quality may suffer or the fee may be high to convert this to an analog format for encoding.

- **MPEG Encoding products** (other than 3DO's MPEGXpress)

There are some excellent MPEG encoding products on the market today, but as of this writing, those that match the quality of 3DO MPEGXpress tend to be more expensive, or tend to produce poorer quality. Also, like most MPEG Encoding services (see previous paragraph), many of these are targeted toward VideoCD MPEG formats.

- **Public domain software encoders**

There are a number of reputedly fairly good non-real-time MPEG encoders available over the Internet. At least one of these, the Berkeley MPEG encoder, even supports rate control. We have not tested this or other encoders, however, so use at your own risk. The Berkeley Web site's URL is:

<http://www-plateau.cs.berkeley.edu/mpeg/index.html>.

B.5 A few words about MPEG-2

This version of 3DO M2 HW does not support MPEG-2 Video decoding.

B.6 Additional Documentation

For more information on the Data Streaming MPEG interface, the "VideoPlayer" MPEG movie example app, and on 3DO M2 Data Streaming in general, see the following documentation:

- The section entitled "Video and Data Streaming" on page 31 of this manual.
- *3DO M2 Video Processing Guide*
- *3DO M2 DataStreamer Programmer's Guide*
- *3DO M2 DataStreamer Programmer's Reference*

Appendix C Man Pages for MPEG Audio Functions

These are man pages for the MPEG audio functions. These pages are included here because they do not appear in the reference documentation.

C.1 CreateMPAudioDecoder

Create resources for a new MPEG Audio decoder.

Synopsis

```
Err CreateMPAudioDecoder( MPADecoderContext **ctx,  
                          MPACallbackFns CallbackFns )
```

Description

Allocate an MPEG audio decoder context structure and initialize its contents. Once the structure is created, you can call MPAudioDecode to start decoding.

Arguments

ctx

Pointer to a MPADecoderContext variable, where a pointer to the MPEG audio decoder context structure can be stored.

CallbackFns

A struct containing two data input callback functions for the MPEG audio decoder. The MPAGetCompressedBfrFn function is called by the decoder to get a compressed MPEG audio buffer to read from, and the MPACompressedBfrReadFn is called when the decoder has finished reading from a compressed data buffer.

Return Value

0 for success, with a pointer to the MPEG audio decoder context structure stored in *ctx, or a negative error code indicating creation errors, e.g. kDSNoMemErr (couldn't allocate memory).

Implementation

Folio call implemented in mpegaudiodecoder folio V30.

Associated Files

<:misc:mpadecoder.h>, System.m2/Modules/mpegaudiodecoder

See Also

DeleteMPAudioDecoder(), MPAudioDecode()

C.2 DeleteMPAudioDecoder

Delete an MPEG Audio decoder context structure.

Synopsis

```
Err DeleteMPAudioDecoder( MPADecoderContext *ctx )
```

Description

Delete an MPEG audio decoder context structure allocated by `CreateMPAudioDecoder`. This function should be called prior to exiting your application and should not be called from the callback functions. Single thread is assumed.

Arguments

`ctx`
Pointer to the MPEG audio decoder context structure to be deleted.

Return Value

0 for success.

Implementation

Folio call implemented in `mpegaudiodecoder folio V30`.

Associated Files

<:misc:mpadecoder.h>, `System.m2/Modules/mpegaudiodecoder`

See Also

`CreateMPAudioDecoder()`, `MPAFlush()`

C.3 MPACompressedBfrReadFn

Done reading the compressed MPEG audio buffer callback function.

Synopsis

```
typedef Err (*MPACompressedBfrReadFn)(const void *theUnit,  
                                       uint8 *buf )
```

Description

Data input callback function called by the decoder when the decoder has finished reading from the compressed data buffer.

Arguments

`theUnit`
Void pointer to the required unit (user data) used by the callback functions.
The decoder will pass this pointer to the callback functions.

`buf`
Pointer to the compressed data buffer that got finished by the decoder.

Return Value

0 for success, with a pointer to the required unit and a pointer to the finished data buffer, or a negative error code indicating errors.

Implementation

Folio call implemented in mpegaudiodecoder folio V30.

Associated Files

<:misc:mpadecoder.h>, System.m2/Modules/mpegaudiodecoder

See Also

MPAGetCompressedBfrFn()

C.4 MPAFlush

Flush input and output pipeline.

Synopsis

```
Err MPAFlush( MPADecoderContext *ctx )
```

Description

Flush the contents of the compressed and decompressed buffers and re-initializes the compressed data buffer struct. This function must be called when branching the input data.

Arguments

ctx
Pointer to the MPEG audio decoder context structure.

Return Value

0 for success or a negative error code for failure.

Implementation

Folio call implemented in mpegaudiodecoder folio V30.

Associated Files

<:misc:mpadecoder.h>, System.m2/Modules/mpegaudiodecoder

C.5 MPAGetCompressedBfrFn

Get a compressed MPEG audio buffer callback function.

Synopsis

```
typedef Err (*MPAGetCompressedBfrFn)(const void *theUnit,  
    const uint8 **buf, int32 *len, uint32 *pts,  
    uint32 *ptsIsValidFlag )
```

Description

Data input callback function called by the decoder to get a compressed MPEG audio buffer from which the data is read. The decoder calls this function when more compressed data is needed during the decoding process.

Arguments

theUnit

Void pointer to the required unit (user data) used by the callback functions.
The decoder will pass this pointer to the callback functions.

buf

Pointer to a buf variable, where a pointer to a compressed data buffer is stored. The decoder reads compressed audio from this buffer.

len The length in bytes of the buffer.

pts Optional presentation time stamp if one is present.

ptsIsValidFlag Boolean flag indicating that the pts value has been set.

Return Value

0 for success, or a negative error code indicating errors. On success, this function returns a pointer to the required unit, a pointer to a buffer containing compressed MPEG audio data, the byte length of the buffer, the PTS and a boolean flag indicating that the PTS value have been set.

Implementation

Folio call implemented in mpegaudiodecoder folio V30.

Associated Files

<:misc:mpadecoder.h>, System.m2/Modules/mpegaudiodecoder

C.6 MPAudioDecode

Decode one frame of MPEG audio.

Synopsis

```
Err MPAudioDecode( void *theUnit, MPADecoderContextPtr ctx,  
    uint32 *pts, int32 *ptsIsValidFlag,  
    uint32 *decompressedBufr, AudioHeader *header )
```

Description

Decode one frame of MPEG audio and store the decoded data in the

decompressedBufr.

Arguments

theUnit

Void pointer to the required unit (user data) used by the callback functions.
The decoder will pass this pointer to the callback functions.

ctx

Pointer to the MPEG audio decoder context structure.

pts

Presentation time stamp indicating the intended time of presentation of the data contained in decompressedBufr. This value is passed in from the callback function.

ptsIsValidFlag

Boolean flag indicating that the presentation time stamp in pts variable has been set.

decompressedBufr

Pointer to a decompressed buffer. This buffer is used by the decoder to store one decompressed MPEG audio frame and size must be BYTES_PER_MPEG_AUDIO_FRAME (refer to <:misc:mpaudiodecode.h>).

header

Pointer to an AudioHeader structure memory used by the decoder to store the header structure from the decoded audio frame. See <:misc:mpeg.h> for the structure definition. The important bits in this header are: mode (mono/stereo/dual channel) and emphasis.

Return Value

0 for success along with the decompressed data stored in decompressedBufr, or a negative error code indicating decoding errors, e.g. MPAUnsupportedLayerErr (MPEG audio layer I and III are not supported).

Implementation

Folio call implemented in mpegaudiodecoder folio V30.

Associated Files

<:misc:mpadecoder.h>, System.m2/Modules/mpegaudiodecoder

See Also

CreateMPAudioDecoder(), DeleteMPAudioDecoder()

Appendix D Examples in Tools Folder

Below is a list of the examples that are shipped with individual tools.

D.1 Audio Examples

D.1.1 test3sf

Source code showing how to call the 3SF sfPlayScore API from a 3DO program.
Located on the CD, inside `":Tools:M2_2.0:Audio:3SF:Examples:"`.

D.2 Development Code Examples

D.2.1 Comm3DOExampleClient 68K

Source code showing how to call the Comm3DO API from a Macintosh program.

Located on the CD, inside

`":Tools:M2_2.0:LowLevel:Comm3DO:Example Client App:"`

D.2.2 Comm3DOExampleClient PPC

Source code showing how to call the Comm3DO API from a Macintosh program.

Located on the CD, inside

`":Tools:M2_2.0:LowLevel:Comm3DO:Example Client App:"`.

D.2.3 C3 Task

Source code showing how to call the Comm3DO API from an M2 program.

Located on the CD, inside

`":Tools:M2_2.0:LowLevel:Comm3DO:Example Client Task:"`.

D.3 Graphics Examples

This is a listing of example and sample code that is to be found in individual tools folders. The path name for these examples is

`":Tools:M2_2.0:Graphics"`.

D.3.1 M2 Font Examples

Shipping in the graphics tools folder with the FontBuilder:

- **DrawTextString <fontName>**
Demonstrates the easiest method of using the font folio to display text strings with the least amount of coding.
- **CreateTextSprite <fontName>**
Demonstrates how to use the font folio to create, render, draw, and dispose a TextSprite.
- **Justification <fontName>**
Demonstrates how to use the font folio to display text using different justification settings.
- **TabStops <fontName>**
Demonstrates how to use the font folio to define tabstops for tab ('\t') delimited text.
- **ShadowText <fontName>**

Demonstrates how to use the font folio to display a series of text strings with a drop shadow.

- **CharacterSpacing <fontName>**

Demonstrates how to use the font folio to vary the blank spacing between characters.

- **WordWrap <fontName>**

Demonstrates how to use the font folio to display word-wrapped text within a specified boundary.

- **TextColor <fontName>**

Demonstrates how to use the font folio to display text in a variety of foreground colors.

- **FloatingText <fontName>**

Demonstrates how to use the font folio to move text about the display.

- **SpinningText <fontName>**

Demonstrates how to use the font folio and the 2d api to give the illusion of a spinning piece of text.

- **UpdateTextInSprite <fontName>**

Demonstrates how to use the font folio to display a block by rendering one word at a time.

- **AsciiTable <fontName>**

Demonstrates how to use the font folio to display the ASCII characters ranging from 0 to 128.

D.3.2 M2 Kanji FontViewer

- **KFontViewer <fontName>**

- This program is new for Release 2.0.

Demonstrates how to use the Kanji font library to load and display S-JIS encoded text on the 3DO system. When loading the font file, the program gives you the option of pre-loading the entire file into memory, or just portions of the file as needed. This option is provided at the start-up screen, as well as is the access time needed to load the font file. The program consists of two examples.

- Example 1: demonstrates how to create and render two sprites containing Hankaku (half-width) characters
- Example 2: demonstrates how to re-use the existing two sprites to render Zenkaku (full-width) characters. In this example each group of Zenkaku characters is listed next to its high-byte S-JIS index value (hex).

Appendix E Examples in Examples Folder

The 3DO M2 CD contains a central “Examples” folder that contains examples for various M2 components. This folder is in the following location:

```
3do_os:M2_2.0:remote:Examples
```

A list follows of all the example programs in the “Examples” folder.

All the example programs are documented by man pages contained in the *3DO M2 Supplemental Portfolio Reference* unless otherwise noted.

Note that some of the locations given in the man pages are incorrect. The locations given in the following list are correct.

E.1 READ THIS NOW!!!

- Makefiles for examples have debugging turned on. For tips and tricks on debugging, see “Tips and Tricks To Set Up Your Program For Debugging” on page 7 of this document.
- Makefiles should include self as a dependency.

E.2 Audio Examples

The audio examples are contained in

```
3do_os:M2_2.0:remote:Examples:Audio
```

E.2.1 Beep

- **tb_envelope** (in Audio:Beep)
Trigger envelopes using the Beep folio.
- **tb_playsamp** (in Audio:Beep)
Play a sample using the Beep Folio.
- **tb_spool** (in Audio:Beep)
Spool audio from memory using the Beep folio.

E.2.2 Juggler

- **tj_canon** (in Audio:Juggler)
Uses the juggler to create and play a semi-random canon.
- **tj_multi** (in Audio:Juggler)
Uses the juggler to play a collection.
- **tj_simple** (in Audio:Juggler)
Uses the juggler to play two sequences.

E.2.3 Misc

- **capture_audio** (in Audio:Misc)
Record the output from the DSPP to a host file.
- **minmax_audio** (in Audio:Misc)
Measures the maximum and minimum output from the DSP.
- **playpimap** (in Audio:Misc)
Automatic rhythm demo that uses lots of AIFF library samples.

- **playsample** (in Audio:Misc)
Plays an AIFF sample in memory using the control pad.
- **simple_envelope** (in Audio:Misc)
Simple audio envelope example.
- **ta_attach** (in Audio:Misc)
Experiments with sample attachments.
- **ta_customdelay** (in Audio:Misc)
Demonstrates a delay line attachment.
- **ta_envelope** (in Audio:Misc)
Tests various envelope options by passing test index.
- **ta_pitchnotes** (in Audio:Misc)
Plays a sample at different MIDI pitches.
- **ta_sweeps** (in Audio:Misc)
Demonstrates adjusting knobs.
- **ta_timer** (in Audio:Misc)
Demonstrates use of the audio timer.
- **ta_tuning** (in Audio:Misc)
Demonstrates custom tuning a DSP instrument.
- **tone** (in Audio:Misc)
Simple audio demonstration.

E.2.4 Score

- **playmf** (in Audio:Score)
Plays a standard MIDI file.
Note that playmf is a shell command and is documented in the chapter of the *3DO Supplemental Portfolio Reference* that describes shell commands.

E.2.5 Sound3D

- **ta_bee3D** (in Audio:Sound3D)
Three sounds in a navigable space using 3DSound API.
See "ta_bee3d" on page 79 of this document for man page.
- **ta_leslie** (in Audio:Sound3D)
Demonstrates directional sound with the 3DSound API.
See "ta_leslie" on page 81 of this document for man page.
- **ta_sound3d** (in Audio:Sound3D)
Simple directional sound using 3DSound API.
See "ta_sound3d" on page 82 of this document for man page.
- **ta_steer3d** (in Audio:Sound3D)
Control a walking man in three-space using the 3DSound API.
See "ta_steer3d" on page 83 of this document for man page.
- **SeeSound** (in Audio:Sound3D:SeeSound)

Sounds in an audio-visual space, using 3DSound and Framework/Pipeline.
See "SeeSound" on page 78 of this document for man page.

E.2.6 SoundEffects

- **sfx_score** (in Audio:SoundEffects)
Uses libmusic.a score player as a sound effects manager.
- **windpatch** (in Audio:SoundEffects)
Creates a "wind" sound effect using the audio folio's patch compiler.

E.2.7 Spooling

- **playsf** (in Audio:Spooling)
Demonstrates how to loop a sound file off disc.
See "playsf" on page 77 of this document for man page.
- **ta_spool** (in Audio:Spooling)
Demonstrates the libmusic.a sound spooler.
- **tsp_algorithmic** (in Audio:Spooling)
Advanced sound player example showing algorithmic sequencing of sound playback.

Known Bugs and Caveats

When running the audio examples `tsp_algorithmic`, `tsp_rooms`, `tsp_spoolsoundfile`, and `tsp_switcher`, you may notice some stuttering and popping. If you see this behavior, try putting the debugger in Special Mode (Command - =). This bug appears to be a result of the hard drive not delivering data fast enough. Special Mode improves I/O speed and is a more accurate representation of actual CD data access.

- **tsp_rooms** (in Audio:Spooling)
Room-sensitive soundtrack example using advanced sound player.
- **tsp_spoolsoundfile** (in Audio:Spooling)
Plays an AIFF sound file from a thread using the advanced sound player.
- **tsp_switcher** (in Audio:Spooling)
Advanced sound player example that switches between sound based on control pad input.

E.3 Event Broker Examples

The Event Broker examples are contained in

```
3do_os:M2_2.0:remote:Examples:EventBroker
```

- **cpdump**
Queries the event broker and prints out a summary of what's connected to the control port.
- **focus**
Talks to the event broker and switches the focus to a different listener.
- **lookie**
Connects to the event broker and reports any events that occur.

- **luckie**
Uses the event broker to read events from the first control pad.
- **maus**
Uses the event broker to read events from the first mouse.

E.4 FileSystem Examples

The FileSystem examples are contained in

`3do_os:M2_2.0:remote:Examples:FileSystem`

- **ls**
Displays the contents of a directory.
- **type**
Type a file's content to the output terminal.
- **walker**
Recursively displays the contents of a directory, and all nested directories.

E.5 Graphics Examples

The Graphics examples are contained in

`3do_os:M2_2.0:remote:Examples:Graphics`

E.5.1 CLT

- **cltspinclip** (in `Graphics:CLT:CLTspinclip`)
Demonstrates various basic CLT features.
See "CLTspinclip" on page 75 of this document for the man page.
Notes
 - Upon exiting the CLTspinclip program a spurious error message is displayed because the program is trying to free a signal that was never allocated. This error does not affect the running of the program and can be ignored. It will be fixed in the next release.
- **cltsurface** (in `Graphics:CLT:CLTsurface`)
Uses the CLT to create and display a 3D surface.
See "CLTsurface" on page 75 of this document for the man page.
- **gsquare** (in `Graphics:CLT:GSquare`)
This example covers: initialization of the graphics folio and gstate, creating and setting bitmaps, double buffering, loading and display of textures, and rendering geometry, including rectangles, strips and fans.
See "GSquare" on page 76 of this document for the man page.
- **height_field** (in `Graphics:CLT:height_field`)
Displays a 3-D terrain section whose texture is determined by its height.
This example is self-documenting. Execute it with no parameters to see usage message.
- **scrolling** (in `Graphics:CLT:Scrolling`)
Demonstrates multi-level parallax scrolling with the CLT.
See "scrolling" on page 77 of this document for the man page.

E.5.2 Fonts

8.12.1.1 READ THIS NOW!!!

You should use CreateM2Make to recreate the makefiles for all the font examples listed here.

8.12.1.2 Fonts Examples

These examples are self-documenting. Execute them with no parameters to see usage message.

- **bounce** (in Graphics:Fonts)
Shows how to set up a TextState, move, scale, and read a TextState's current position by bouncing the text around the screen.
- **colors** (in Graphics:Fonts)
Shows how to change text colors.
- **newfont** (in Graphics:Fonts)
Runs through a series of font folio tests, and times all the tests.
- **rotate** (in Graphics:Fonts)
Shows how to rotate text on screen.
- **showfont** (in Graphics:Fonts)
Display characters of a font.
- **showmessage** (in Graphics:Fonts)
Outputs any words on the command line of the debugger to the display monitor.
- **testn1** (in Graphics:Fonts)
Tests whether the font folio correctly handles \n characters.

E.5.3 Frame

- **anim** (in Graphics:Frame:anim)
Displays object with keyframe animation using Framework extensions.
See "anim" on page 75 of this document for the man page.
- **freespun** (in Graphics:Frame:freespun)
Freely rotate specified objects within a binary SDF.
See "freespun" on page 76 of this document for the man page.
- **m2perf** (in Graphics:Frame:m2perf)
Program to measure the performance of Framework, Pipeline, and the M2
- **newview** (in Graphics:Frame:newview)
Rotate specified objects within a binary SDF using control pad.
See "newview" on page 76 of this document for the man page.
- **sceneperf** (in Graphics:Frame:sceneperf)
Indicates performance data for rendering a scene.
- **testspin** (in Graphics:Frame:testspin)
Rotate objects within a binary SDF and record performance statistics.

This example is self-documenting. Execute it with no parameters to see usage message.

E.5.4 Frame2d

- **automapper** (in Graphics:Frame2d:Automapper)
Demonstrates how to map the corners of a sprite.
- **drawlines** (in Graphics:Frame2d:DrawLines)
Demonstrates three methods of drawing lines.
- **movesprite** (in Graphics:Frame2d:Movesprite)
Shows how to move, scale, and rotate a sprite
- **points** (in Graphics:Frame2d:Points)
Shows how to draw points.
- **rectangles** (in Graphics:Frame2d:Rectangles)
Demonstrates two methods of drawing rectangles.
- **renderorder** (in Graphics:Frame2d:RenderOrder)
Shows how to alter the render order of sprites using a list-based approach.
- **simplesprite** (in Graphics:Frame2d:SimpleSprite)
Very simple example of how to draw a single sprite.
- **spin3d2d** (in Graphics:Frame2d:spin3d2d)
Shows how to display 3D and 2D graphics simultaneously.

E.5.5 GraphicsFolio

- **autosizeview** (in Graphics:GraphicsFolio)
Illustrates how to create a Bitmap and View automatically sized for the prevailing video mode.
- **basicview** (in Graphics:GraphicsFolio)
Illustrates how to create a basic View.

E.6 Kernel Examples

- **defaultport**
Demonstrates using a task's default message port to communicate with a child thread.
- **fasttiming**
Demonstrates how to use the high accuracy kernel timing services.
- **memdebug**
Demonstrates the memory debugging subsystem.
- **metronome**
Demonstrates how to use the metronome timer feature.
- **msgpassing**
Demonstrates sending and receiving messages between two threads.
- **signals**
Demonstrates how to use signals.

-
- **timerread**
Demonstrates how to use the timer device to read the current system time.
 - **timersleep**
Demonstrates how to use the timer device to wait for an amount of time specified on the command-line.

E.7 Miscellaneous Examples

The Miscellaneous examples are contained in

`3do_os:M2_2.0:remote:Examples:Miscellaneous`

- **compression** (in `Miscellaneous:Compression`)
Demonstrates use of the compression folio.
- **dbgconsole** (in `Miscellaneous:DbgConsole`)
Example program used to demonstrate 3DODebug functionality.
- **numinfo** (in `Miscellaneous:DebugExamples`)
Example program used to demonstrate 3DODebug functionality.

E.8 MPEG Examples

The MPEG examples are contained in

`3do_os:M2_2.0:remote:Examples:MPEG`

- **photo** (in `MPEG:photo`)
This example application shows how to use the MPEG-Video decoder interface for still-image decompression.
The example loads an MPEG-Video still-image (I-frame) from a file into M2 memory. The file format is simply an MPEG-1 Video elementary stream containing only an I-picture. (It can be created, for example, using Sparkle.) The Photo app then opens the MPEG-Video device for I-picture-only decoding, so that the device allocates no reference-frame buffers in main memory. The device returns the decompressed I-frame to the Photo app as soon as it is finished decoding, without the pipeline delay incurred with full I/P/B movie decoding.
- **playfast** (in `MPEG:playfast`)
Demonstrates the power of the 3DO M2 MPEG-1 Video decoder, and also serves as example code for the MPEG-1 Video device driver.
Works by loading into 3DO M2 RAM as much of an input MPEG-1 Video elementary stream as it has space for, and then playing this stream repeatedly in a loop.
This example does not use the Data Streaming interface to M2's MPEG-1 Video decoder.

E.9 Data Streaming Examples

This section summarizes and gives pointers to the Data Streaming example applications. For more detail and notes on most of these applications, see "Data Streaming Examples" on page 37.

The Data Streaming examples are contained in

3do_os:M2_2.0:remote:Examples:Streaming

- **DataPlayer** (in Streaming:DataPlayer)
Uses the DATA subscriber to “play” a stream containing DATA chunks. The application simply prints statistics about data blocks as they are delivered, but it exemplifies how to set up, interact with, and destroy the DATA subscriber.
- **EZFlixPlayer** (in Streaming:EZFlixPlayer)
Plays an EZFlix (3DO M2 SW-decodable video) stream, with or without a synchronized native 3DO M2 audio stream.
- **PlaySA** (in Streaming:PlaySA)
Plays one or more native 3DO M2 audio streams woven into a single multiplexed stream.
- **VideoPlayer** (in Streaming:VideoPlayer)
Plays an MPEG-1 Video stream, with or without a synchronized native 3DO M2 audio stream.

Appendix F Selected Man Pages for Examples

This appendix contains man pages for examples whose man pages do not appear in the *3DO M2 Supplemental Portfolio Programmer's Reference*.

F.1 anim

Description

Given a binary SDF file with keyframe animation data and the name of the top level object within the file, load the object and animate it. Use the control pad to rotate, translate, and zoom the object about each axis.

Usage:

```
anim [InitGP params] file.csf <objectName>

example: anim skeleton.csf skeleton_world
```

F.2 CLTspinclip

Demonstrates various basic CLT features

Arguments

none

Description

CLTspinclip demonstrates the fundamentals of CLT use by putting a spinning icosahedron on the screen. It also uses special code to create oversized bitmaps, allowing hardware clipping.

D-pad, L-shift, R-shift:	rotate icosahedron
A button+D-pad left/right:	set filtering mode to bilinear or point
A button+D-pad up/down:	turn transparency on/off
B button:	zoom in
C button:	zoom out
Stop button:	quit

Location

examples/graphics/clt/cltspinclip

F.3 CLTsurface

Uses the CLT to create and display a 3D surface

Arguments

none

Description

CLTsurface shows how the CLT can be used to generate 3D graphics. It demonstrates a number of features, including lighting and video feedback

D-pad, L-shift, R-shift:	rotate surface
A button+D-pad up/down:	zoom in/out
A button+D-pad left/right:	turn lighting on/off
B button+D-pad:	change number of triangles used to draw surface
C button+D-pad left/right:	turn video feedback on/off
C button+D-pad up/down:	turn autorotation on/off
Start button:	warp surface
Stop button:	quit

Location
examples/graphics/clt/cltsurface

F.4 freespin

Description

Given a binary SDF file and the name of an object within the file, load the object and display it. Rotate the object freely about the X and Y axis without requiring user intervention. Control pad events will stop and exit the program.

Usage:

```
freespin [InitGP params] file.csf [objectName]
```

F.5 GSquare

Demonstrates basic CLT use

Arguments
none

Description

GSquare is an introductory example showing how to use the CLT to draw triangles onto the screen. It draws, and moves around, both textured and untextured triangles. Pressing any control pad button causes it to quit.

Location
examples/graphics/clt/gsquare

F.6 newview

newview

Description

Given a binary SDF file and the name of an object within the file, load the object and display it. Use the control pad to rotate, translate, and zoom the object about each axis.

Usage:

```
newview [InitGP params] file.csf [objectName]
```

F.7 playsf

Play, pause and stop playback of a soundfile from disk.

Format

```
playsf [filename]
```

Description

This program plays back a sound file from disk using a background thread. The foreground task monitors the control pad; you can pause, restart and stop playback using the play/pause and stop buttons respectively. The example uses Messages to send commands from the foreground task to the background thread.

Arguments

<filename>

File name of an AIFF file. If you don't provide one, the example file "spA.aiff" in the MarkovMusic(@) directory will be used.

Associated Files

spA.aiff

Caveats

The control pad handling function uses GetControlPad with wait enabled, which means the foreground task sits and waits for button presses even after the soundfile has finished playing and the background thread has politely cleaned up after itself. You need to hit the Stop button to terminate the program.

Location

Examples/Audio

F.8 scrolling

Demonstrates multi-level parralax scrolling with the CLT

Arguments

none

Description

Scrolling demonstrates 11 layers of full screen parallax scrolling at 60 fps.

Start button: pause

A button: resume

Stop button: quit

Location

examples/graphics/clt/scrolling

F.9 SeeSound

SeeSound

Sounds in an audio-visual space, using 3DSound and Framework/Pipeline.

Format

```
seesound [reverb [cuelist]]
```

Description

This program positions three objects in a three-dimensional space, and allows the user to move about in the space using the control pad. The space is rendered graphically using the Framework/Pipeline API, and sonically using the 3DSound API. By varying the command line arguments, the user can hear the relative effect of the various 3DSound cues on the three-dimensional illusion.

To move around using the control pad:

Up

moves you forward

Down

moves you backward

Left

rotates you anticlockwise

Right

rotates you clockwise

A

repositions you under the "bee".

Two of the objects are stationary: a large rectangular block with an associated looped "clonking" sample and a large cylinder, with an associated synthetic telephone sound. The third, a synthetic bee represented as a small cone, does a random walk confined to a predetermined distance from its initial location.

All the geometry for this example is done with Pipeline matrix operations, and then mapped into the coordinate system of the 3DSound library.

Note: to use this example, you first need to assign the alias "samples" to your General MIDI Sample Library, and then run the shell script "makepatch.script" in the directory Examples/Audio/Sound3D to build the set of patches loaded by the program.

Arguments

reverb

A floating point number from 0.0 to 1.0, determining the global reverberation amount in the final mix. The amount of sound sent to the reverb is independently calculated for each sound, based on the distance of the sound from the observer, and the overall wet/dry mix is controlled with this parameter. By default, the value is 0.05. The higher the number, the harder it becomes to accurately determine the position of a distant sound.

cuelist

a string of digits selected from the list below. Including a digit in the string enables the corresponding cue. The default cuelist is equivalent to the string 1347.

1

Interaural Time Delay

2

Interaural Amplitude Difference

3

Pseudo-HRTF Filter

4

Distance-squared rule for amplitude attenuation

5

Distance-cubed rule for amplitude attenuation

7

Doppler shift. Note that the dopplering sounds a little strange, due to the lack of momentum in movement - effectively velocity changes are instantaneous, and hence so are frequency changes.

Associated Files

Sound3D/patches/bee3d.mp, Sound3D/patches/bee3dspace.mp,
Sound3D/patches/clonk.mp, Sound3D/patches/phone.mp,
Sound3D/makepatch.script

Location

Examples/Audio/Sound3D/SeeSound

See Also

ta_bee3d

F.10 ta_bee3d

Three sounds in a navigable space using 3DSound API.

Format

ta_bee3d [reverb [cuelist]]

Description

This program positions three sounds in a three-dimensional space, and allows the user to move about in the space using the control pad. The sounds are positioned and moved using the 3DSound functions in the music library. By varying the command line arguments, the user can hear the relative effect of the various 3DSound cues on the three-dimensional illusion.

To move around using the control pad:

Up
 moves you forward
Down
 moves you backward
Left
 rotates you anticlockwise
Right
 rotates you clockwise
A
 repositions you under the "bee" sound.

Two of the sounds are stationary: a looped "clonking" sample and a synthetic telephone. The third, a synthetic bee, does a random walk confined to a predetermined distance from its initial location.

Note: to use this example, you first need to assign the alias "samples" to your General MIDI Sample Library, and then run the shell script "makepatch.script" to build the set of patches loaded by the program.

Arguments

reverb

A floating point number from 0.0 to 1.0, determining the global reverberation amount in the final mix. The amount of sound sent to the reverb is independently calculated for each sound, based on the distance of the sound from the observer, and the overall wet/dry mix is controlled with this parameter. By default, the value is 0.05. The higher the number, the harder it becomes to accurately determine the position of a distant sound.

cuelist

a string of digits selected from the list below. Including a digit in the string enables the corresponding cue. The default cuelist is equivalent to the string 134.

- 1
 Interaural Time Delay
- 2
 Interaural Amplitude Difference
- 3
 Pseudo-HRTF Filter
- 4
 Distance-squared rule for amplitude attenuation
- 5
 Distance-cubed rule for amplitude attenuation
- 7
 Doppler shift

Associated Files

patches/bee3d.mp, patches/bee3dspace.mp, patches/clonk.mp,
patches/phone.mp, makepatch.script

Location

Examples/Audio/Sound3D

See Also

SeeSound(@)

F.11 ta_leslie

Demonstrates directional sound with the 3DSound API.

Format

```
ta_leslie [backamp [cuelist]]
```

Description

This example badly simulates a leslie, an electroacoustic sound processor commonly used with electric organs. The simulation routes a source sound (an organ sample) through two 3D Sound objects positioned back to back some distance apart from each other in front of the observer and rotating about their mutual center.

The speed of rotation can be controlled with the control pad.

Up

fast rotation

Down

slow rotation

Note: to use this example, you first need to assign the alias "samples" to your General MIDI Sample Library, and then run the shell script "makepatch.script" to build the set of patches loaded by the program.

Arguments

backamp

A floating point number between 0.0 and 1.0, supplied as the tag argument TAG_S3D_BACK_AMPLITUDE to the function Create3DSound().

cuelist

a string of digits selected from the list below. Including a digit in the string enables the corresponding cue. The default cuelist is equivalent to the string 147.

1

Interaural Time Delay

2

Interaural Amplitude Difference

3

Pseudo-HRTF Filter

4

- Distance-squared rule for amplitude attenuation
- 5 Distance-cubed rule for amplitude attenuation
- 7 Doppler shift
- 8 Smooth amplitude changes

The last option, "smooth amplitude changes", makes a marked improvement in the sound.

Associated Files

patches/ta_leslie.mp, makepatch.script

Location

Examples/Audio/Sound3D

F.12 ta_sound3d

Simple directional sound using 3DSound API.

Format

```
ta_sound3d [backamp [cuelist]]
```

Description

This example rotates a sound through a complete circle over a five-second time span. It is intended to show the effect of the tag TAG_S3D_BACK_AMPLITUDE and the flag S3D_F_SMOOTH_AMPLITUDE on a 3DSound object.

Note: to use this example, you first need to assign the alias "samples" to your General MIDI Sample Library, and then run the shell script "makepatch.script" to build the set of patches loaded by the program.

Arguments

backamp

A floating point number between 0.0 and 1.0, supplied as the tag argument TAG_S3D_BACK_AMPLITUDE to the function Create3DSound().

cuelist

a string of digits selected from the list below. Including a digit in the string enables the corresponding cue. The default cuelist is equivalent to the string 14.

- 1 Interaural Time Delay
- 2 Interaural Amplitude Difference

-
- 3 Pseudo-HRTF Filter
 - 4 Distance-squared rule for amplitude attenuation
 - 5 Distance-cubed rule for amplitude attenuation
 - 7 Doppler shift
 - 8 Smooth amplitude changes

Associated Files

patches/test3dsound.mp, makepatch.script

Location

Examples/Audio/Sound3D

F.13 ta_steer3d

Control a walking man in three-space using the 3DSound API.

Format

ta_steer3d [reverb [cuelist]]

Description

With this program, you steer the sound of a walking man around a plane using the control pad. By varying the command line arguments you can hear the effect of various cues on the three-dimensional illusion.

The "man" starts moving at a velocity of about 1 m/s from where you're standing straight forward. Each press on the arrow keys changes his velocity. The control pad buttons work as follows:

- Up
 - Add 1 m/s in the forward direction
- Down
 - Add 1 m/s in the backward direction
- Left
 - Add 1 m/s in the leftward direction
- Right
 - Add 1 m/s in the rightward direction
- Stop
 - Stop walking, play a sound at the current position
- Play
 - Start walking again, with a 1 m/s forward velocity
- A
 - End the program

With a little adept button pressing, you can make the man walk by your shoulder, or around you in a circle.

Note: to use this example, you first need to assign the alias "samples" to your General MIDI Sample Library, and then run the shell script "makepatch.script" to build the set of patches loaded by the program.

Arguments

reverb

A floating point number from 0.0 to 1.0, determining the global reverberation amount in the final mix. For example, a value of 0.25 means that 0.25 of the output signal for a given sound comes from the reverb and 0.75 from the dry signal. The overall amplitude is controlled by the distance of the sound from the listener.

cuelist

a string of digits selected from the list below. Including a digit in the string enables the corresponding cue. The default cuelist is equivalent to the string 147.

- 1 Interaural Time Delay
- 2 Interaural Amplitude Difference
- 3 Pseudo-HRTF Filter
- 4 Distance-squared rule for amplitude attenuation
- 5 Distance-cubed rule for amplitude attenuation
- 7 Doppler shift

Associated Files

patches/sound3dspace_nice_stereo.mp, patches/walking.mp,
makepatch.script

Location

Examples/Audio/Sound3D



Getting Started With 3DO M2 Release 2.0

Version 2.0– May1996

Copyright © 1996 The 3DO Company and its licensors.

All rights reserved. This material constitutes confidential and proprietary information of The 3DO Company. This documentation is subject to a license agreement with The 3DO Company and may be used only by parties to such agreement. Use by any other persons, and/or for any purpose not expressly authorized by the agreement, is strictly prohibited.

3DO's LICENSOR(S) MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. 3DO'S LICENSOR(S) DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME JURISDICTIONS. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

3DO and the 3DO Logos are trademarks and/or registered trademarks of The 3DO Company. ©1996 The 3DO Company. All rights reserved. Other brand or product names are the trademarks or registered trademarks of their respective holders.

Contents

Preface

About This Document.....	GSG-v
How This Document Is Organized	GSG-v
Typographical Conventions	GSG-vi

1

3DO M2 Development System

Overview.....	GSG-1
3DO Development Environment	GSG-1
.....	GSG-2
3DO M2 Developer's Console System	GSG-2
3DO M2 Developer Documentation Set 2.0	GSG-3
3DO M2 Portfolio and Toolkit Release Version 2.0 CD-ROM	GSG-5
Macintosh Environment.....	GSG-5
Hardware	GSG-5
Software	GSG-5
Optional Additions to the Development Environment	GSG-6
Graphics Development	GSG-6
Audio Development	GSG-6

2

3DO M2 Development Card Installation

About the 3DO M2 Development Card.....	GSG-8
Installing the 3DO M2 Development Card.....	GSG-8
Cabling the 3DO M2 Development Card	GSG-10
Installing the M2 2.0 Software.....	GSG-11

3

Installing the Software

Before you Install the Software	GSG-14
Finding the Online HTML Documentation Files	GSG-14
Using 411 Help files	GSG-15
Installing MPW	GSG-15
Installing the 411 Help files	GSG-15
Installing 3DO M2 2.0 Software	GSG-16
Locating the 3DO Software	GSG-19
Troubleshooting Your Installation	GSG-20
Problems Installing System Extensions	GSG-20

4

Building a 3DO M2 Application

An Overview of the Build Process	GSG-21
The Makefile	GSG-23
A Sample Program	GSG-23
Creating a Makefile with CreateM2Make	GSG-24
Enabling the Debugger in the MakeFile	GSG-25
Building the Program	GSG-25
Launching and Configuring 3DODebug	GSG-26
Running the newview Program	GSG-28
Summary	GSG-29

Preface

About This Document

This *Getting Started With 3DO M2 Release 2.0* guide shows you how to install the M2 development card, the M2 2.0 software, and how to run an example program using MPW and 3DODebug.

How This Document Is Organized

This document is organized as follows:

- ◆ **Chapter 1, “3DO M2 Development System,”** provides a general overview of the 3DO development environment, lists the components of the 3DO M2 Console System, and describes some of the features of the 3DO M2 Development Card, version G. This chapter also lists the contents of the Developer Documentation Set and the contents of the 3DO M2 Portfolio and Toolkit Release Version 2.0 CD-ROM.
- ◆ **Chapter 2, “3DO M2 Development Card Installation** describes how to install and cable the 3DO M2 Development Card, Revision G in your Macintosh.
- ◆ **Chapter 3, “Installing the Software,”** tells you what software you must install to use the 3DO M2 Development Card, version G. It steps you through installing the 3DO M2 2.0 software and gives you information about how to use 411 Help files and online documentation.
- ◆ **Chapter 4, “Building a 3DO M2 Application,”** provides an overview of the application build process and describes how to use CreateM2Make, an MPW shell script, to create a makefile, and run an example program.
- ◆ **Appendix A, “File Formats,”** describes the file formats commonly used for storage on 3DO CD-ROM disc, within the system itself, and for art and music development on a 3DO M2 development system.

Typographical Conventions

The following typographical conventions are used in this book:

Item	Example
code example	<code>Item CreateRenderBuffer(int rb_Size)</code>
procedure name	<code>ModifyGraphicsItem()</code>
new term or emphasis	In M2, <i>characters</i> are objects that can be displayed on the screen.
file or folder name	The <i>remote</i> folder, the <i>demo.scr</i> file.

3DO M2 Development System

Overview

The 3DO M2 Development System is the hardware and software environment used in conjunction with the Apple Macintosh to produce M2-based titles for the 3DO M2 Customer Console System.

This chapter describes the components of the 3DO development environment. It includes the following topics:

Topic	Page
Overview	1
3DO Development Environment	1
Macintosh Environment	5
Optional Additions to the Development Environment	6

3DO Development Environment

The 3DO M2 development system includes the following :

- ◆ The 3DO M2 Developer's Console
- ◆ The 5-volume 3DO M2 Documentation Set 2.0
- ◆ 3DO M2 Portfolio and Toolkit Release Version 2.0 CD-ROM
- ◆ Diab Data Compiler (C/C++) 3.7a, or later
- ◆ Link3DO, version 1.0.15
- ◆ 3DODebug 2.1a13

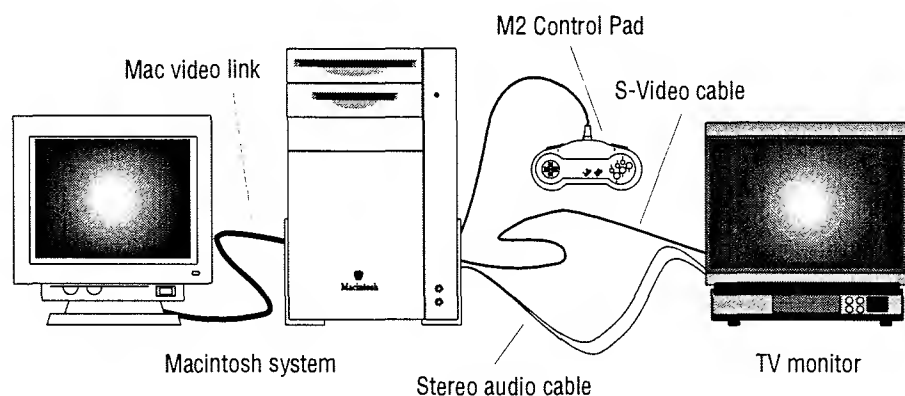


Figure 1-1 *Components of the 3DO M2 Developer Environment*

3DO M2 Developer's Console System

The 3DO M2 Developers Console System consists of the following:

- ◆ 3DO M2 Development Card, (version G for Release 2.0), including a slot for PCMCIA peripherals
- ◆ 3DO 6-button Control Pad prototype
- ◆ Breakout cable
- ◆ One serial cable

3DO M2 Development Card, version G Specification Highlights

The 3DO M2 Development Card, version G is the basic card required for 3DO title development in the M2 environment. This card is a single NuBus card, sufficient for Audio and Video programming for 3DO M2 titles, utilizing M2-specific tools, or doing code development.

The 3DO M2 Development Card includes the following features.

- ◆ 10 Custom Processors
- ◆ CPU type: Power PC 602
- ◆ 528 Megabytes/secondary memory bus bandwidth
- ◆ 100 Million pixels/second rendering speed
- ◆ 1 Million polygons/second peak rendering speed
- ◆ 4 MBytes SDRAM minimum
- ◆ 64-bit memory bus
- ◆ 640X480 and 320X240 at 24-bit or 16-bit color depths
- ◆ MPEG-1 video built-in
- ◆ Built-in Graphics Features
 - Mip-Mapping
 - Z-Buffering
 - Perspective Correction
 - Gourand Shading

Alpha Channel Support

- ◆ Built-in Audio Features
 - ◆ 32-channels of sound
 - ◆ DSP with 2:1 hardware audio decompression

3DO M2 Developer Documentation Set 2.0

The documentation set exists in 5 volumes of ring binders.

Table 1-1 *Volume 1 of 3DO M2 Developer Documentation 2.0*

<i>3DO M2 Release 2.0 Global Table of Contents</i>	Provides a detailed Table of Contents for the entire Developer's Documentation Set.
<i>3DO M2 Release 2.0 Global Index</i>	Provides a detailed Index for the entire Developer's Documentation Set.
<i>3DO M2 Release 2.0 Release Notes</i>	Last minute information that has not been included in the documentation set. See the Release Notes for changes/updates/or important bug fixes in the release.
<i>Getting Started With 3DO M2 Release 2.0</i>	Describes the 3DO M2 Development System, how to install hardware and software , and build a 3DO M2 application.
<i>3DO M2 Debugger Programmer's Guide</i>	Describe how to use the 3DO M2 Debugger to monitor and control C, C++, and assembly language programs running on the 3DO M2 Development Card.
<i>3DO M2 DataStreamer Programmer's Guide</i>	Describes how to combine audio and graphics files to operate in real time synchronization.
<i>3DO M2 DataStreamer Reference</i>	Describes how to convert sound and graphics files from their original formats into a form usable by M2
<i>3DO M2 Command List Toolkit</i>	How to write 3DO M2 graphics programs at the register level.
<i>3DO M2 Video Processing Guide</i>	Overview of how to incorporate video into your games. Includes material on Quick-Time, Video tape, and AIFF files.
<i>3DO M2 FontBuilder User's Guide</i>	Describes how to use the M2 FontBuilder to create anti-aliased fonts for the 3DO M2 system.
<i>3DO M2 CD-ROM Mastering Guide</i>	Tutorial instructions for building a game CD, complete with an application checklist

:

Table 1-2 *Volume 2 of 3DO M2 Developer Documentation*

<i>3DO M2 Graphics Tools</i>	Describes how to use the MPW graphic tools including data streaming.
<i>3DO PostPro 2.0 User's Guide</i>	File conversion into 3DO M2 formats.
<i>3DO M2 Graphics Programmer's Guide</i>	Descriptions of how to use Graphics Frameworks, Pipeline, Folio and other graphic components of 3DO M2.
<i>3DO M2 Graphics Programmer's Reference</i>	Complete set of function calls and parameters for graphics, in alphabetical order.

Table 1-3 *Volume 3 of 3DO M2 Developer Documentation*

<i>3DO M2 Tools for Sound Design</i>	Provides tutorial-style and reference information to all 3DO Sound Designer tools, plus troubleshooting and reference information.
<i>3DO M2 Audio and Music Programmer's Guide</i>	Overviews, programming tutorials, sample code, and function call definitions of the audio and music folios.
<i>3DO M2 Audio Programmer's Reference</i>	All of the function calls and parameters of the audio and music calls, in alphabetical order.

Table 1-4 *Volume 4 of 3DO M2 Developer Documentation*

<i>3DO M2 Portfolio Programmer's Guide</i>	Describes how to develop code for the 3DO M2 operating system. Includes material on queues, task schedules, and I/O.
<i>3DO M2 Portfolio Programmer's Reference</i>	Function calls and parameters for the 3DO M2 operating system calls.

:

Table 1-5 *Volume 5 of 3DO M2 Developer Documentation*

<i>DIAB Data D-C++ Language User's Manual</i>	How to use the compiler, including all command line options.
<i>DIAB Data PowerPC Target User's Manual</i>	Describes the target configuration specifics and all target-dependent command line options.

Table 1-5 Volume 5 of 3DO M2 Developer Documentation

<i>DIAB Data D-AS/PowerPC Assembler User's Manual</i>	Describes the D-AS™ assembler; its directives , expressions and macros
<i>DIAB Data Utilities User's Manual</i>	A collection of utilities to do file archives, display profile data from program runs, and dump files.

3DO M2 Portfolio and Toolkit Release Version 2.0 CD-ROM

There are nine folders at the top level of this CD-ROM. The folders and their contents are:

- 3do_os**-contains the runtime portion of the development system
- Examples**-Event Broker, Audio Folio, Debugger
- Streaming**-data streaming libraries and sample code
- 3dodebug**-contains the 3DO M2 Debugger
- docs**-online version of the 5-volume documentation set
- roms**-contains ROM images and tools to download them into your computer.
- cdrommaster**-contains the software necessary to build a stand alone CD
- Interfaces**-contains C header files for the 3DO M2 Application Program Interface.
- tools**-stand alone applications, MPW tools, and 3DO M2 tools for audio, graphics, the Diab Data Compiler, and Link3DO.

Macintosh Environment

The following hardware and software components are prerequisites for using the 3DO M2 Development System.

Hardware

- ◆ Any Macintosh 68040, or PowerPC with an unused Nubus slot.
- ◆ At least 16MB of RAM, 24 MB is recommended
- ◆ A video monitor with composite video input (RCA-type phono plug) or S-video (S-VHS) input (4-pin mini DIN)
- ◆ Video cable (S-Video or composit video)

Software

- ◆ System 7.5 or later, or Kanjitalk 7.5.1, or later
- ◆ MPW version 3.2 or later. Version 3.4, or later is recommended for Power Mac users to take advantage of native tools.
- ◆ QuickTime 2.0, or later

Caution: Be sure to turn off virtual memory on your Macintosh while the 3DO M2 Development Card is installed on your system. Virtual memory is not compatible with the 3DO Debugger. Note also that the 3DO Debugger requires 32-bit addressing; do not use 24-bit addressing mode.

Optional Additions to the Development Environment

The additions listed in this section are for developers working in specialized environments.

Graphics Development

2D drawing packages

3D modeling packages

Audio Development

Digidesign Audio Media II (16-bit) sound card

Sound Designer II (sample editor)

Any MIDI sequencer

MIDI keyboard

Audio mixer and amplifier

Mac MIDI Interface

3DO M2 Development Card Installation

This chapter describes how to install the 3DO M2 Development Card, Revision G, in your Macintosh. It covers the following topics:

Topic	Page
About the 3DO M2 Development Card	7
Installing the 3DO M2 Development Card	8
Cabling the 3DO M2 Development Card	10
Installing the M2 2.0 Software	11

Caution: *You may damage the 3DO or Macintosh hardware if you don't follow the proper installation procedures.*

About the 3DO M2 Development Card

The 3DO M2 Development Card enables you to create 3DO titles and applications on your Macintosh, and can be installed in any Macintosh with a NuBus slot. The Revision G Development Card you have received has three connectors: audio/control pad, composite video and S-video (S-VHS), all located on the rear connector panel. This is a departure from previous revisions of the development card, which used a four-connector rear panel. If you are upgrading from a previous development card, please pay particular attention to the cabling section.

The Revision G Development Card is shipped with the following items:

- ◆ one six-button control pad (not shipped with upgrades)
- ◆ *Read This First 3DO M2 Development Card* (this document)
- ◆ composite video cable (not shipped with upgrades)
- ◆ audio and control pad breakout cable

Installing the 3DO M2 Development Card

This section describes how to install the 3DO M2 Development Card in your Macintosh. If you haven't recently installed a NuBus Card, read the section on installing NuBus cards in the Macintosh User's Guide that came with your system. NuBus card installation varies on different Macintosh systems.

1. Turn off the Macintosh and monitor.
2. Remove the Macintosh CPU cover.
3. Ground yourself to prevent damage to the card.

Warning: *Static electricity can damage the 3DO M2 Development Card. Always ground yourself prior to physical contact with the card. Refer to the Macintosh documentation for the grounding procedure recommended when installing NuBus cards. As a minimum precaution, touch one hand to the power supply to ground yourself.*

4. Remove the Development Card from its static-proof bag.
5. Align the Development Card's NuBus connector with the matching connector in your Macintosh and push the card down until it is securely seated (see Figure 2-1). You must push down firmly to seat the card in the NuBus slot, but don't force it. If the insertion doesn't feel right, remove the card and check the connector for bent pins or other damage.

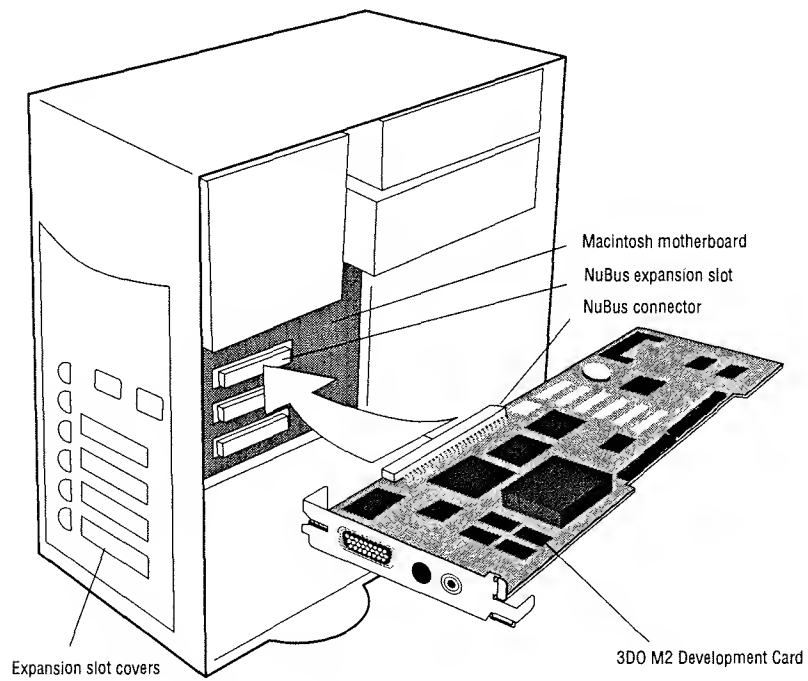


Figure 2-1 Installing the 3DO M2 Development Card.

Cabling the 3DO M2 Development Card

Connect the Development Card as follows:

1. Ensure that your Macintosh is turned off.
2. Locate the breakout cable (see) and connect the end with the 26-pin D-type connector to the corresponding connector on the Development Card's rear panel (see).

Note: If you are upgrading from a previous version of the development card, do not attempt to connect the older card's control port cable to the Revision G development card.

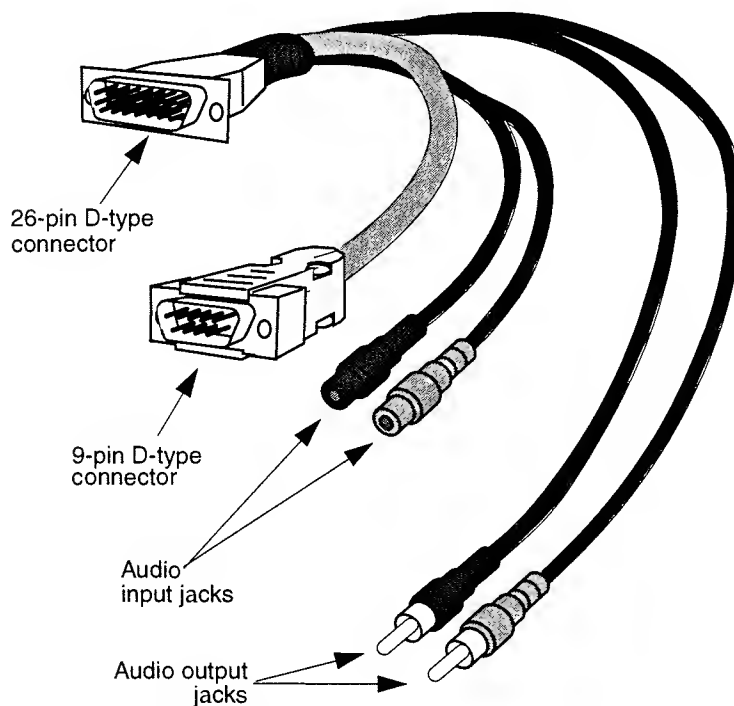


Figure 2-2 Breakout cable.

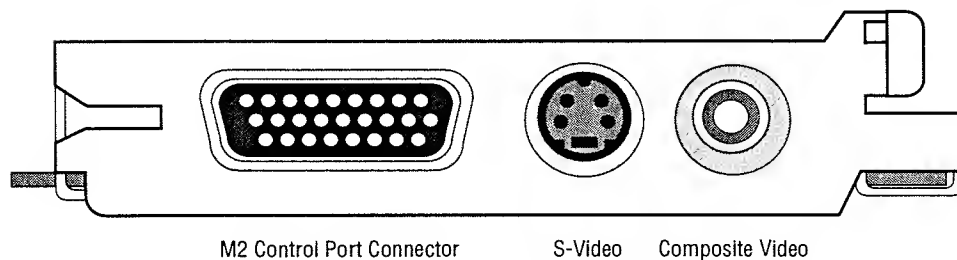


Figure 2-3 Revision G Development Card connector panel.

3. Connect the breakout cable's audio output jacks to the corresponding connectors on either the TV monitor or on a stereo system.

If you have an external sound source, you can connect it to the development card by connecting the external source's output jacks to the audio input jacks on the breakout cable.

4. Attach the breakout cable's 9-pin D-type connector to the control pad.
5. Attach either the composite video cable or S-video cable to the corresponding connector on the Development Card and the TV monitor.
6. Once the cables are attached, turn on your system and monitor.

Note: *If the TV monitor causes interference with the Macintosh monitor, separate them. If the cable provided is too short, you can extend it with a standard RCA male-to-female stereo cable.*

Installing the M2 2.0 Software

In order to use the 3DO Development Card, you must install the 3DO M2 2.0 software, located on the 3DO M2 2.0 CD-ROM included with the documentation. Software installation is covered in Chapter 3, "Installing the Software."

Installing the Software

This chapter lists the software you need to install to use the new 3DO M2 Development Card. It tells you how to install the 3DO M2 2.0 Software. The 3DO M2 2.0 Software consists of the 3DO M2 operating system, tools, examples, and other files. The installation process is modular, with most of the installation performed by the Apple Installer.

This chapter covers the following topics:

Topic	Page
Before you Install the Software	14
Installing MPW	15
Installing 3DO M2 2.0 Software	20
Troubleshooting Your Installation	20

Before you Install the Software

The 3DO M2 Release 2.0 includes three types of online documentation:

- ◆ 411-Help, for use within the MPW environment
- ◆ HTML formatted manual pages which can be used with either the Netscape Communications' Netscape Navigator™ or NCSA Mosaic browsers.
- ◆ An ASCII version of the online manual pages which can be viewed with any text editor.

These files are located in the *docs:M2_2.0* folder on the CD.

Release 2.0 of the 3DO M2 Developer's Documentation Set provides the operating system folio calls in Hypertext Markup Language (HTML) format as a supplement to the hard copy documentation. In addition, these calls are also included as 411 Help files.

The HTML documentation is viewable with NCSA Mosaic and Netscape Communications' Netscape Navigator™. Of the two, Netscape Navigator is recommended. You can download these browsers from the 3DO InfoServer at (415) 261-3405 (see "3DO Infoserver's InfoBase" on page 43). The path is *3DO InfoServer Desktop/User Info/User Library*.

Note: You do not need an Internet connection, MacTCP, or any networking software to view the Online documentation.

You need to install the 411-Help file, *PortfolioHelp* into your MPW environment in order to use these files.

Finding the Online HTML Documentation Files

The HTML documentation is included on your 3DO M2 2.0 CD in the *docs* folder. Inside this folder is an *html* folder that contains a file called *Portfolio_TOC.html*. If you have Netscape installed on your system, double clicking on this file will start Netscape and bring you to the index of all folio calls.

If you don't have Netscape installed, or if you are using another browser, you should use your browser to open and view this file. To open the file:

- ◆ Select Open Local from the File menu in NCSA Mosaic, or
- ◆ Select Open File from the File menu in Netscape Navigator.

Find the *docs* folder and open the file *docs:M2_2.0:Portfolio_TOC.html*. You can now access the complete online documentation set.

Using 411 Help files

The 411 Help files provide easy access to online function descriptions. 411 Help files are not installed automatically by the installer.

In order to use 411 Help, you need to have the 411 Help tool and the appropriate help files available.

- ◆ Version 3.2 or later of MPW includes 411 Help, but 411 Help is only installed if you select the 411 Help check box in the MPW installation dialog box. To use the 3DO help files, you only need to install the 411 Help tools (the complete 411 Help installation package includes reference files for using MPW and the Macintosh operating system, which are not required).
- ◆ There is a single 411 Help file on the M2 2.0 CD-ROM. This file is called *PortfolioHelp*, and is located in the *docs:M2_2.0:411* folder.

Installing MPW

To install MPW, follow the steps in your MPW documentation set.

Note: Remember to select the 411 Help check box in the MPW installation dialog box so that 411 Help tools are installed.

Installing the 411 Help files

After you have installed the MPW 411 Help tools, you need to install the 3DO Help file and configure it as follows:

1. Move *PortfolioHelp*, located in the *docs:M2_2.0:411* folder into the folder you designated as the 411 Help folder.
2. Open MPW.
If you have just installed 411 Help, you will be prompted to locate your 411 folder.
3. Select the appropriate folder.
4. From the 411 menu, choose Set 411 Help files and select the folder in which all 411 Help files are located.

This updates 411 Help so it recognizes the newly installed files.

5. From the 411 menu, select Set First File.

You should see *PortfolioHelp* listed in the dialog box.

6. Select *PortfolioHelp* and click OK.

You are placed on the Overview page of *PortfolioHelp*.

Installing 3DO M2 2.0 Software

To install the 3DO M2 2.0 Software, follow these steps:

1. Insert the CD-ROM into your Macintosh CD-ROM drive.
2. Double-click on the CD-ROM icon.

The 3DO window opens, showing the 3DO M2 Software folders you see in Figure 3-1.

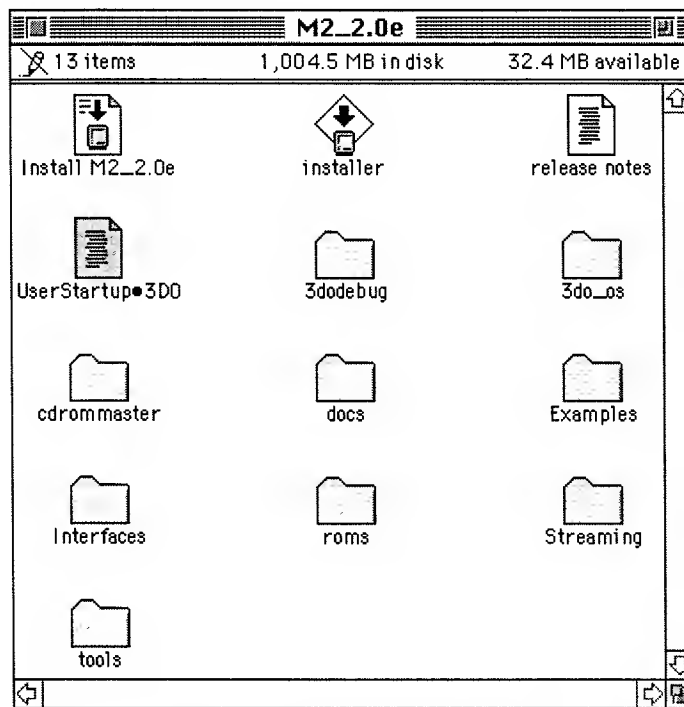


Figure 3-1 3DO M2 Software folders

3. Double click on the *Install M2_2.0* file to open the main install window shown in Figure 3-2.

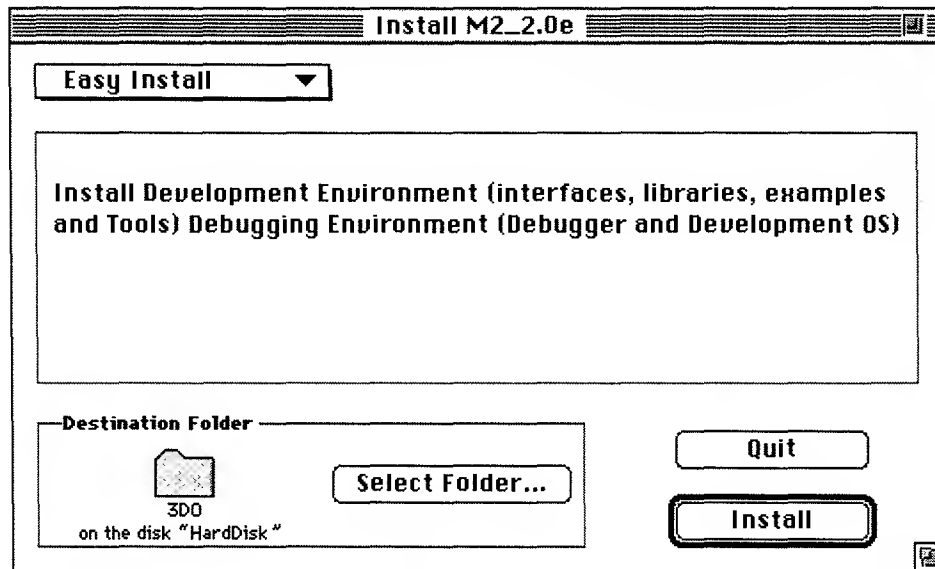


Figure 3-2 The main Install window displays.

4. Select the installation type from the pop-up menu at the top left of the Install dialog box. The available choices are:

Select	To...
Easy Install	Install everything in the development environment.
Custom Install	Check the appropriate box to install either the development station software or Debugger station software.

5. Click on the Select Folder button shown in Figure 3-3 to select the destination disk and folder for your installation.

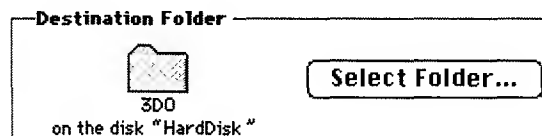


Figure 3-3 Default 3DO Destination Folder

If you do not specify a location, the 3DO folder shown in Figure 3-3 is created at the root of the boot hard disk.

6. Click the Install button shown in Figure 3-2 when you are ready to begin the installation.

The standard install icons you see in Figure 3-4 display until the install completes.

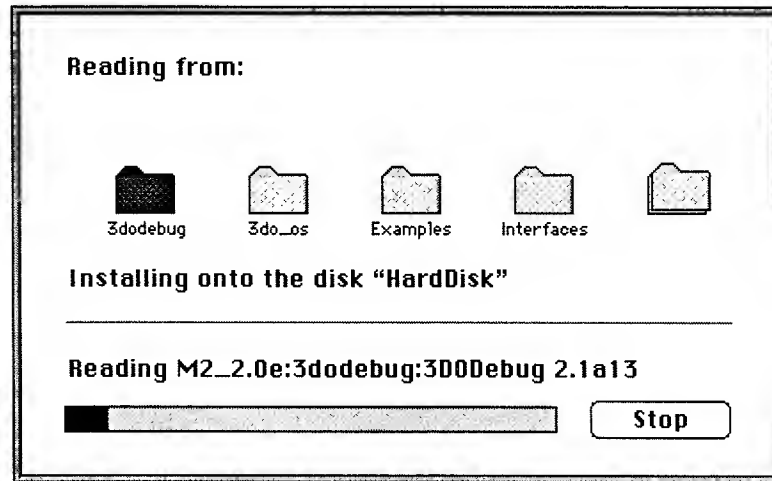


Figure 3-4 *Install progress icons*

7. Click the Quit button shown in Figure 3-5 when the installation is complete.

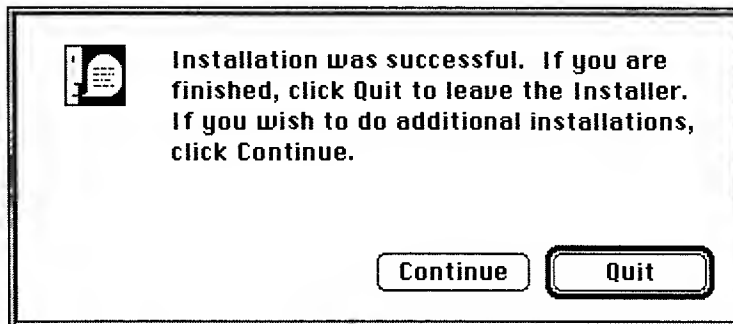


Figure 3-5 *Successful Install*

8. Drag the MPW file *UserStartup•3DO* from the CD (See Figure 3-1), to the MPW folder on your hard drive.

The installer records where the 3DO software was placed; the new *UserStartup•3DO* file uses that location to set the MPW environment variables.

Note: *If you rename an old version of UserStartup•3DO, make sure that the new name doesn't begin with "UserStartup." By convention, MPW executes all files that begin with UserStartup. Executing two or more copies of UserStartup•3DO will result in an unusable development environment.*

9. Open your System Folder and delete the file:

System Folder:Extensions:3DO:3DO_Setup



Figure 3-6 *Delete old 3DO_Setup file from your System folder.*

10. Locate the following two files on the CD-ROM and drag them onto your system folder:

M2_2.0:tools:M2_2.0:Audio:Sound Manager 3.1:Sound Manager

and

M2_2.0:tools:M2_2.0:Video: EZFlixQT:EZFlix

Note: *The Sound Manager and EZFlix files are not installed with your 3DO M2 Software installation. If you are not going to be using either EZFlix or Sparkle, then omit step 10.*

11. Restart your Macintosh.

This Sound Manager and the EZFlix QT components are now enabled.

Locating the 3DO Software

Once you've got MPW and the 3DO software installed, you need to set MPW to recognize the 3DO installation and your program files.

If MPW cannot locate the 3DO software, the dialog box in Figure 3-7 displays.

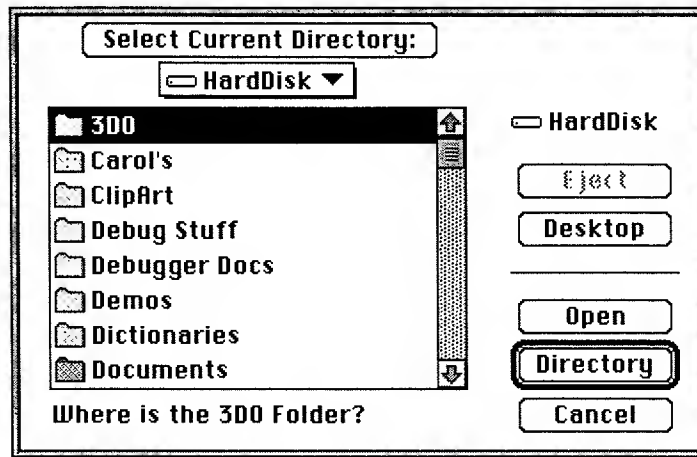


Figure 3-7 Locating the 3DO Directory containing the 3DO folder which contains the 3DO Installation.

To locate the 3DO M2 software

1. Navigate so that the path `:3DO:Examples:3DO` displays in the box under the Select Current Directory: box on the dialog box shown in Figure 3-7.
2. Click on the Select Current Directory button.

Troubleshooting Your Installation

This section briefly lists some troubleshooting information for your installation.

Problems Installing System Extensions

If you have a system extension installed that you are replacing with a newer version—for example, if you are installing QuickTime 2.0—that extension is considered in use and you are informed that you cannot install it.

To install the newer version, move the older version into the Trash, move the newer version into the *Extensions* folder (inside the *System Folder*), and restart your machine. You can then empty the Trash to have only one version (the latest and correct version) on your system.

Building a 3DO M2 Application

This chapter describes the steps involved in building a 3DO M2 application. Special attention is paid to the CreateM2Make script used to create a “makefile” which drives the compilation and linking process. The information presented here assumes you have correctly installed MPW and the M2 3DO Operating System.

This chapter covers the following topics:

Topic	Page
An Overview of the Build Process	21
Creating a Makefile with CreateM2Make	24
Building the Program	25
Running the newview Program	28

An Overview of the Build Process

The process of building a 3DO M2 application is much the same as writing any program and getting it to run on the target hardware. Your source files are compiled into object files which are then run through a linker (Link3DO), linked with the appropriate libraries, and made executable. If you wish, dynamic-link libraries (DLLs) are available to enhance memory-intensive operations (DLLs can be included during compile and link but don’t actually get executed until needed at runtime). For more information about DLLs, see the *3DO M2 Portfolio Programmer’s Guide*. Add the customary debugging iterations to the build process and you’ll soon find all the elements of your application working in harmony to produce a winning title. See Figure 4-1 for a diagram of the build process.

To build a 3DO M2 application, follow these steps and refer to Figure 4-1.

1. Use 3DO Debug to access the DIAB Data compiler to write your C source files.
2. Run your source files through CreateM2Make.
3. Use MPW to build your application.
4. Use 3DO Debug to debug and/or execute your application.

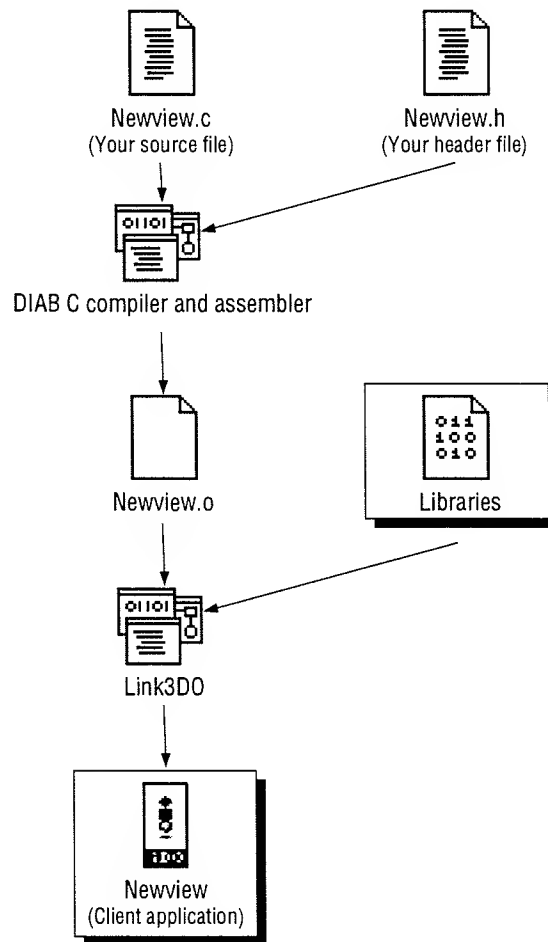


Figure 4-1 Build process

The Makefile

The commands that drive the build process are collected in a *makefile* created using a script called CreateM2Make. When you use the Build command in MPW, the makefile automatically invokes the compiler and linker to compile and link your applications. You only need to create a makefile once. However, you may edit it and revise it many times throughout the process of building a 3DO M2 application.

A Sample Program

This example of compiling a program uses the *newview* sample program shown in the Figure 4-1 build process and contained on the 3DO M2 CD-ROM.

The first step in the compilation is to supply MPW with the pathname to the *newview* folder. You can do this in any of three ways:

- ◆ Type the explicit pathname to the *newview* file in your worksheet and leave it there for future reference. Later, when you want to start building the application, you can click on that line and press Enter.
- ◆ Choose Set Directory from the Directory menu in MPW. When the standard file dialog box appears, locate the directory where *newview* is located (note that only folders, not files, are shown) and click the Select Current Directory button.

If you have not moved any files since installation, *newview* should be located in *3DO:Examples:M2_2.0:Graphics:Frame:newview*.

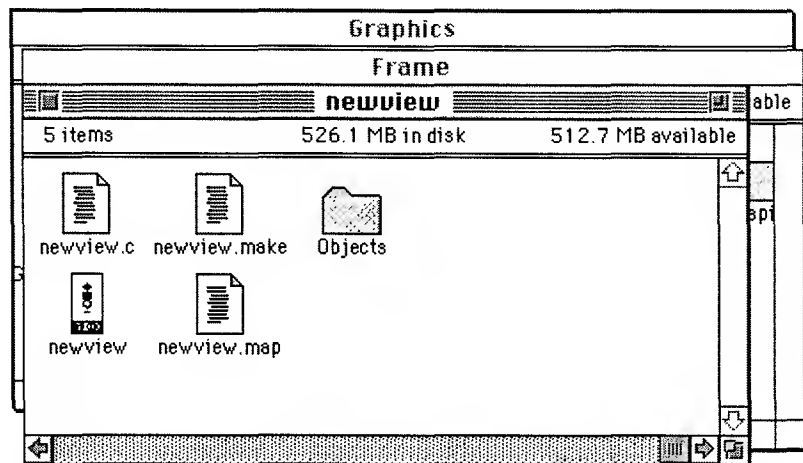


Figure 4-2 Path to *newview* program

- ◆ If MPW isn't running, you can always navigate to the *newview* folder in the Finder and double-click *newview.c*. This will launch MPW and automatically set the Directory to the current directory. It will also open a second window for the text file *newview.c*, which you can close.

Creating a Makefile with CreateM2Make

CreateM2Make is an MPW shell script for creating a basic makefile, sufficient for linking the appropriate libraries, and resolving references to DLLs. As you become more expert, you will probably want to make improvements to the resulting makefile. You will definitely want to read the debugger documentation as well as the *3DO M2 Portfolio Programmer's Guide*.

Now that the directories are set in MPW, you can create a makefile.

To create a makefile, in the MPW shell, type:

```
createm2make newview newview.c
```

and press Enter or Command+Return. MPW creates a file in the *newview* folder called *newview.make*

Note: *This example only uses one source file. For projects with more than one source file, feel free to list additional files on this line.*

This command uses *newview.c* as input to create a final executable program called *newview*. (If you type *createm2make* without any arguments, CreateM2Make supplies the syntax.)

The makefile is intimately related to MPW and to program management in general. CreateM2Make creates a text file that defines:

- ◆ Relationships between C source files and the object code that is generated
- ◆ Libraries needed for linking
- ◆ How to set up the compiler for doing that kind of compilation
- ◆ How to resolve DLLs
- ◆ How to copy resultant compiled code into the */remote* folder so you can run it

You can edit *newview.make* to put in more libraries, replace the different object files with ones that you create, and so on. You can also add dependencies. For example, CreateM2Make doesn't take into account which files include other ones.

Newview.make is not a script in that it doesn't actually do anything. Rather, it is a set of rules for creating the *newview* application—not the instructions themselves.

Enabling the Debugger in the MakeFile

By default, the MakeFile produced by CreateM2Make generates a program that can be debugged. When you want to build a release version of your program, you will want to turn on the compiler optimizations and disable debugging (for details, see the *3DO M2 Debugger Programmer's Guide*).

Building the Program

Once you have a makefile, you can tell MPW to build the program by following these steps:

1. From the Build menu, choose Build... (or press Command-B)
A dialog box displays with a request for a program name.
2. Type in the program name without an extension as you see in Figure 4-3.

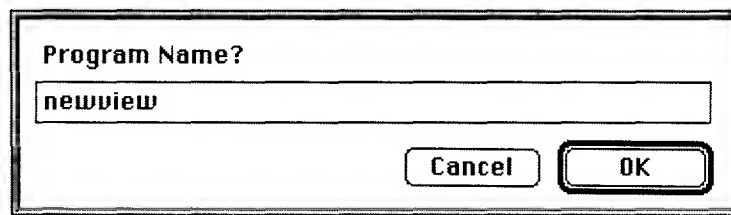


Figure 4-3 *Build program name*

3. Click OK

The progress of the build displays as you see in Figure 4-4.

```

HardDisk:MPW:Worksheet
MPW Shell
newview: make
newview: mop
creates2acka newview newview.c

• 11:37:33 AM ----- Build of newview
• 11:37:33 AM ----- Analyzing dependencies.
• 11:37:34 AM ----- Executing build commands.
Set Echo 0
• ----- Linking
link3do -r -D -Htime=now -Hsubsys=1 -Hname=newview -Htype=5 -Hstack=32768 -o newview :objects:newview.c.o -L"HardD
Set Echo 0
• ----- Setting up in remote folder
setfile newview -c '3D00' -t 'PROJ'
duplicate newview -y HardDisk:300:300_DS:M2_2.0e:Remote:
Set Echo 0
• ----- Creating the debugging script
echo "set sourcedir" > "HardDisk:300:300_DS:M2_2.0e:Remote:newview.spt"
directory
echo "set sourcedir" > "HardDisk:300:300_DS:M2_2.0e:Remote:newview.spt"
echo "set datadir" > "HardDisk:300:300_DS:M2_2.0e:Remote:newview.spt"
echo "set datadir" > "HardDisk:300:300_DS:M2_2.0e:Remote:newview.spt"
• 11:37:33 AM ----- Done
  
```

Figure 4-4 MPW worksheet display

To monitor the build progress, look at the status pane in the upper left corner of the MPW window shown in Figure 4-4. The normal display in the pane is *MPW Shell*. The contents of this pane changes to reflect the status of the various executions. Compilation and linking is complete when the status line returns to *MPW Shell*.

Another way to monitor build progress is to see what happens to the various files by watching the Finder update the folders where these changes take place. If you want to monitor the build process this way, *before* clicking OK, (in the above procedure) move your MPW window out of the way, switch to the Finder and open the *newview* folder and the *remote* folder and place them side by side.

Launching and Configuring 3DODebug

To launch 3DODebug, for the first time after installation, follow these steps. Otherwise go directly to step 3.

1. Double-click the 3DODebug icon.

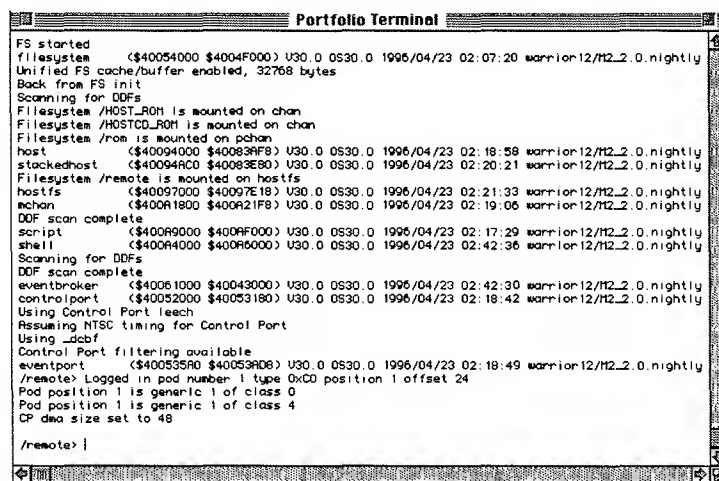


Figure 4-5 Portfolio Terminal Window

As the Debugger loads information scrolls in the Portfolio Terminal window that you see in Figure 4-5.

2. Press ENTER when the load finishes to display the Debugger prompt.

/remote

3. Go to the Target Menu and select Setup.

Set the values in the Target Setup dialog box that you see in Figure 4-6.

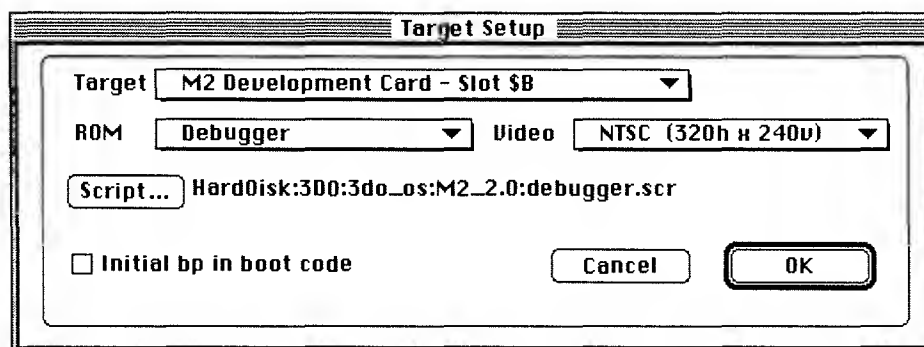


Figure 4-6 Target Setup to locate 3DO M2 Development Card

Setting the ROM to the 3DO Debugger means that the 3DO Operating System will be downloaded from the Macintosh to the M2 Development card. This keeps the development card always loaded with the current version of the OS (instead of what is in Flash ROM).

For a description of other options and more information, see the 3DO M2 Debugger Programmer's Guide.

4. Click OK.

When launch is complete, you'll see a couple of lines indicating the pad position state and the 3DODebug prompt appears:

```
/remote>
```

Note: This document assumes you are launching 3DODebug for the first time after installation. If you have previously launched the debugger, choose *HW Reset* from the *Target* menu.

Running the *newview* Program

You are now ready to run the *newview* program, which is located on your hard drive in `3DO:3DO_OS:M2_2.0:remote:examples:graphics:frame:newview`.

To run the *newview* program, you need to change the working directory that you are in.

1. Enter the following command at the debugger prompt to change the directory:

```
/remote> cd examples/graphics/frame/newview
```

2. Drag the file *textcow.csf* into the same working directory.

3. Type the command line

```
debug newview textcow.csf
```

This command displays the cow in Figure 4-7 on your TV monitor.

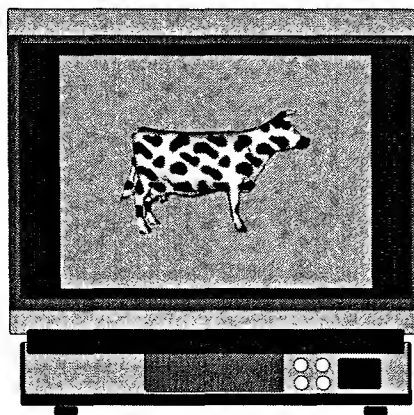


Figure 4-7 Test cow example

Substitute *mipcube.csf* for *texcow.csf* and return to Step 2, if you want to display a black and white checked cube instead of the cow.

With the cow or cube now displayed on the TV monitor connected to your M2 development card (it won't display on your Mac), you can use the 3DO M2 control pad to rotate, move or zoom the model.

Summary

Working through this chapter you've seen how to set up MPW, create a makefile with CreateM2Make, build the application from the source and libraries, and run it. These essential steps are all it takes to develop applications for M2. Of course there's more to it than that. Chapter 5 provides some pointers about where to read the details about programming for M2.

File Formats

A number of different file formats are used by developers when programming the 3DO M2. These formats are described in their respective manuals, as listed in the following table:

Table E-1 *File Formats*

File Format	Location
IFF File Format	3DO M2 Portfolio Programmer's Guide
3DO Score File Format	3DO M2 Tools for Sound Design
3DO M2 Texture File Format	3DO M2 Graphics Tools
Texture Formats	3DO M2 Graphics Programmer's Guide
Bitmaps	3Do M2 Graphics Programmer's Guide
Scene Description Format	3DO M2 Graphics Programmer's Guide



3DO M2 Debugger Programmer's Guide

Version 2.0—May 1996

Copyright © 1996 The 3DO Company and its licensors.

All rights reserved. This material constitutes confidential and proprietary information of The 3DO Company. This documentation is subject to a license agreement with The 3DO Company and may be used only by parties to such agreement. Use by any other persons, and/or for any purpose not expressly authorized by the agreement, is strictly prohibited.

3DO's LICENSOR(S) MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. 3DO'S LICENSOR(S) DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME JURISDICTIONS. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

Other brand or product names are the trademarks or registered trademarks of their respective holders.

Contents

Preface

About the 3D0 Debugger.....	DBG-vii
System Requirements	DBG-viii
Contents of This Book	DBG-viii
Typographical Conventions	DBG-viii

1

Introducing the 3D0 M2 Debugger

About the 3D0 M2 Debugger	DBG-2
Setting Your System Up for Debugging.....	DBG-2
Configuring Your Source Directories	DBG-4
Configuring Your Application's Makefile	DBG-5
Setting Debugger Preferences.....	DBG-6
Special Mode	DBG-6
Using the Target Setup Dialog Box	DBG-7
Starting the M2 Debugger	DBG-8
Using the Portfolio Terminal Window	DBG-9
Setting Up Your Source and Symbolic-Information Files	DBG-10
Specifying File Locations with the Directories!Setup Dialog Box	DBG-11
Specifying the Location of Your Application's Source Files	DBG-12
Specifying the Location of Your Application's Symbolic-Information File	DBG-13
Starting a Debugging Session	DBG-13
Setting Up the Debugger for the newview Program	DBG-14
Two Ways to Supply the Location of a File	DBG-15
Executing the debug Command	DBG-16
Using the Source Window	DBG-18
The Disassembly Window	DBG-19

Specifying Source Directories with an .spt File	DBG-22
What's an .spt File?	DBG-22
Summary	DBG-23

2

Using the Debugger

Resuming a Debugging Session	DBG-26
Using BreakPoints	DBG-27
Using the BreakPoints Window	DBG-29
Stepping Through Code	DBG-29
Using the Step In Command	DBG-30
Using the Step Over Command	DBG-31
Working with Variables	DBG-32
Features of the Variables Window	DBG-33
Working with the Variables Window	DBG-33
Two Ways to Use the Variables Window	DBG-34
Examining the Value of a Variable, Method 1	DBG-34
Examining the Value of a Variable, Method 2	DBG-35
Changing the Value of a Variable	DBG-36
Working with an Array	DBG-38
Examining Data	DBG-40
Using the Data Window	DBG-40
Using the Memory Dump Window	DBG-42
Using the Task Window	DBG-43
Viewing and Modifying the Power PC Registers	DBG-43
The Power PC User Registers Window	DBG-44
The Power PC FP Registers Window	DBG-45
The Power PC Supervisor Registers Window	DBG-46
The Stack Crawl Window	DBG-47
The Triangle Engine Disassembly Window	DBG-48
Summary	DBG-50

A

Menu Reference

The File Menu	DBG-52
New	DBG-52
Open (Command + O)	DBG-52
Close (Command +W)	DBG-52
Save (Command + S)	DBG-52

Directories	DBG-53
Page Setup	DBG-53
Print (Command + P)	DBG-53
.Special Mode (Command + =)	DBG-53
Quit (Command + Q)	DBG-54
The Edit Menu.....	DBG-54
Undo	DBG-54
Cut (Command + X)	DBG-54
Copy (Command + V)	DBG-54
Paste	DBG-55
Clear	DBG-55
Select All (Command + A)	DBG-55
Preferences	DBG-55
The View Menu.....	DBG-55
Disassembly (Command + J)	DBG-56
Data (Command + B)	DBG-56
Source (Command + K)	DBG-56
Variables (Command + M)	DBG-56
Symbols (Command + Y)	DBG-56
BreakPoints (Command + U)	DBG-56
Terminal (Command + T)	DBG-57
Task	DBG-57
Status	DBG-57
PPC User Registers	DBG-57
PPC FP Registers	DBG-57
PPC Supervisor Registers	DBG-57
Stack Crawl	DBG-57
Triangle Disassembly	DBG-57
Send Variable Data (Command + D)	DBG-58
The Execution Menu	DBG-58
Go (Command + G)	DBG-58
Stop (Command + .)	DBG-58
Kill Stopped Task	DBG-58
Step Over (Command + ,)	DBG-58
Step In (Command + ;)	DBG-59
Ignore DebugBreakPoint	DBG-59
Ignore BreakPoints	DBG-59
Initial BreakPoint in task.	DBG-59

The Target Menu.....	DBG-59
Setup	DBG-59
HW Reset (Command + H)	DBG-59
Memory Dump	DBG-60
Ignore Printf (Command + /)	DBG-60
The Tools Menu.....	DBG-60
FileSystem Trace	DBG-60
CD Access Log	DBG-60
Index	DBG-61

Preface

This book is a guide to using the 3DO M2 debugger, a Macintosh application that lets you monitor and debug C and assembly-language programs running under the M2 development system.

This Preface briefly describes the M2 debugger, outlines the system requirements for using it, and lists the contents of this book and the typographical conventions used in the text. The major topics covered in this Preface are:

Topic	Page
About the 3DO Debugger	vii
About the 3DO Debugger	vii
Contents of This Book	viii
Typographical Conventions	viii

About the 3DO Debugger

The M2 debugger allows you to run and step through programs as they are being executed. The debugger also lets you set, use, and change breakpoints, and lets you examine CPU and custom ASIC (BDA) registers during the debugging phase of application development.

More specifically, these are some of the things that the M2 debugger allows you to do:

- ◆ Download and execute object code using the M2 development card
- ◆ Execute and step through M2 applications as you run them in real time
- ◆ Set and use software breakpoints

- ◆ Display and edit CPU registers, custom registers, and memory locations
- ◆ Monitor and change the values of variables
- ◆ Perform source-level and assembly-level debugging
- ◆ Reset the M2 development card and restart the operating system.

System Requirements

To use the debugger, you should have access to a working 3DO M2 development system and to an Apple Power Macintosh or Macintosh Quadra running Operating System 7.5 or higher. You must also have access to a 3DO M2 system installed in accordance with the installation procedures described in the *3DO Development Environment Installation Guide*.

Contents of This Book

This book contains the following chapters and appendixes:

- ◆ Chapter 1, "Introducing the 3DO M2 Debugger," introduces the 3DO M2 debugger and describes its features.
- ◆ Chapter 2, "Using the Debugger," presents a tutorial that shows in more detail how to debug an M2 program.
- ◆ Appendix C, "Menu Reference," lists and explains the menu commands you can use during an M2 debugging session.

Typographical Conventions

The following typographical conventions are used in this book:

Item	Example
code example	<code>Scene_GetStatic(scene)</code>
procedure name	<code>Char_TotalTransform()</code>
new term or emphasis	In M2, <i>characters</i> are objects that can be displayed on the screen.
file or folder name	The <i>remote</i> folder, the <i>demo.scr</i> file.

Introducing the 3DO M2 Debugger

The M2 debugger is the keystone of the 3DO M2 development environment. It is used not only to debug M2 applications, but to run them in the Macintosh development environment. In fact, the 3DO debugger is routinely used as a user interface for the M2 system during both the creation and debugging stages of application development.

This chapter introduces the 3DO M2 debugger—also known as 3DODebug—and shows you how to prepare for a debugging session. When you finish this chapter, you'll be ready to move on to Chapter 2, "Using the Debugger," which shows how to use 3DODebug to debug M2 applications.

This chapter covers the following topics:

Topic	Page
About the 3DO M2 Debugger	2
Setting Your System Up for Debugging	2
Setting Debugger Preferences	6
Starting the M2 Debugger	8
Setting Up Your Source and Symbolic-Information Files	10
Starting a Debugging Session	13
Specifying Source Directories with an .spt File	22
Summary	23

About the 3DO M2 Debugger

The 3DO M2 debugger environment has two components:

- ◆ The 3DO M2 debugger, named *3DODebug*, which runs on the Macintosh.
- ◆ The *target monitor program*, which resides on the 3DO M2 development card.

The target monitor program supports debugging by managing low-level software tasks that are required during the debugging process.

Figure 1-1 shows how 3DODebug and the target monitor program work together during the debugging phase of application development.

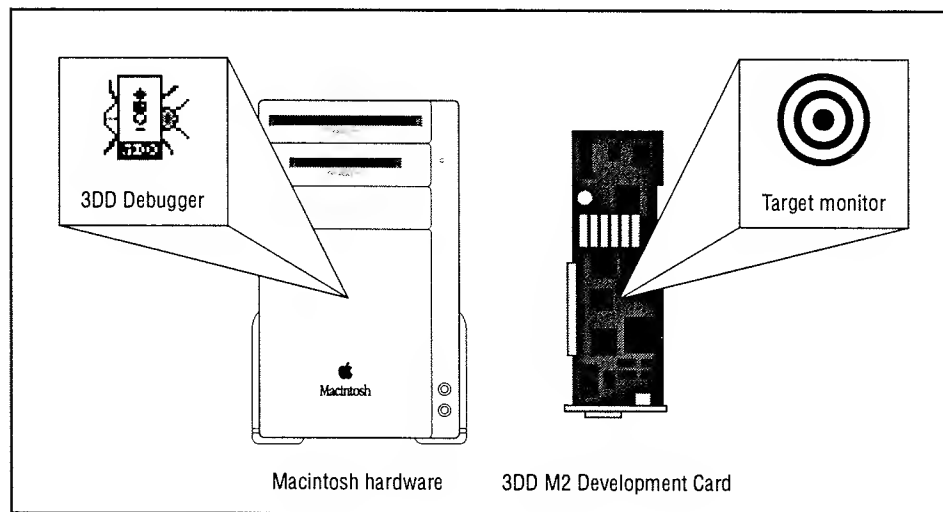


Figure 1-1 The 3DO debugger environment.

Setting Your System Up for Debugging

This section explains how to set up the M2 system for a debugging session. It explains how to launch the debugger for the first time, so if there is a debugger preferences (*3DODebug.Prefs*) file present, delete it before proceeding.

Also, before you start the M2 debugger, ensure that all software and hardware is properly installed and that the monitor connected to your developer (dev) card is turned on.

To demonstrate the configuration of the M2 debugger and the rest of the M2 debugging environment, this chapter uses an application named *newview* that is provided on your distribution CD-ROM. You will also use the *newview* program in Chapter 2, "Using the Debugger."

The *newview* program has one source file, named *newview.c*. You can find that file in the `:3DO:Examples:Graphics:Frame:newview` folder on your M2 distribution CD-ROM.

Before you start setting up your system for a debugging session, you should build the *newview* application. To build the program, follow the steps provided the *Getting Started With 3DO M2 Release 2.0* manual. But be sure to build the program with debugging enabled, as described in "Building a Debugging Version of an Application" on page 5.

When you build and execute the *newview* application using a Macintosh and an M2 development card, the program displays a model of a cow on the video monitor that is connected to your dev card. With the M2 game pod, you can rotate the cow, change the angle from which it is viewed, and move it toward the camera and away from the camera.

Figure 1-2 shows the output of the *newview* program.

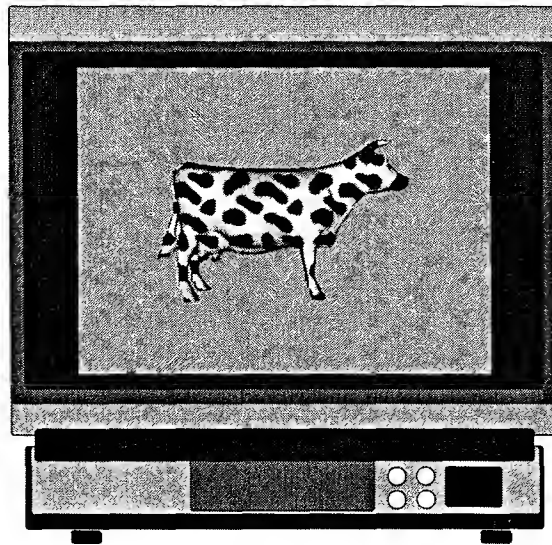


Figure 1-2 *Output of the newview program.*

Configuring Your Source Directories

The first step in preparing for an M2 debugging session is to make sure that the directory structure you will be using for your application's source files is set up properly. When you install the M2 software, the Installer places the source files that make up the *newview* project in a directory named

`3DO:Examples:M2_2.0:Graphics:Frame:newview`

Figure 1-3 illustrates the default contents of this directory.



Figure 1-3 *The newview application's directory structure.*

Note: Before starting the debugger, you might need to increase the amount of memory the Macintosh allocates to 3DODebug. Use the *Get Info ...* command in the Finder to check memory allocation.

As Figure 1-3 shows, the *newview* folder contains:

- ◆ An MPW makefile named *newview.make*. In the 3DO development environment, a makefile is a file that is used to build an application. If you need to refresh your memory on what makefiles are and how to use them, see your MPW documentation. You will be working with a makefile in this chapter, so it is important to have a basic understanding of makefiles before you start learning about how to debug M2 applications.
- ◆ The *newview* file. This is the executable file for the *newview* application. To build an executable, you use a makefile. For more details about using makefiles to build executables, see the *Getting Started With 3DO M2 Release 2.0* manual and your MPW documentation.
- ◆ A C-language source file named *newview.c*. This is the source file from which the *newview* program is built.

- ◆ An *Objects* folder in which MPW stores the interim object files that the M2 compiler and linker use during the build process.

Configuring Your Application's Makefile

The makefile that builds the *newview* application is the *newview.make* file shown in Figure 1-3 on page 4. This makefile, in the form in which it is shipped to you on your distribution CD-ROM, builds a debugging version of the *newview* program.

For the exercises presented in this chapter, you should use the debugging version of the *newview.make* file, just as it is provided on the 3DO distribution CD-ROM. But if you wanted to modify the file to build a release version of the *newview* program, that would be an easy change to make. The following paragraphs explain how to modify the supplied *newview* makefile to build a debugging version or a release version of the *newview* application.

Building a Debugging Version of an Application

To generate a debug version of an M2 program, a makefile must enable debugging by using the `-g` build option, which places a special kind of symbolic information in the program's executable. This option is used in the sample makefile that is shipped with the *newview* program. Near the top of the *newview.make* file, notice the following lines:

```
# Choose one of the following:
DEBUG_OPTIONS = -g # For best source-level debugging
# DEBUG_OPTIONS = -XO -Xunroll=1 -Xtest-at-bottom -Xinline=5
```

The second of these three lines—the one that contains the `-g` option—enables debugging by including symbolic information in your executable. But notice that the third line is commented out with the `(#)` symbol. Commenting that line out turns off optimizations that are not needed in debugging.

Because these three lines are preset to generate debugging information, you should leave them the way they are during the debugging phase of application development.

Building a Release Version of an Application

Later, when you finish debugging your application, you can modify your makefile to generate a release version of your title by following just two steps:

1. Add a comment symbol `(#)` to the second line shown in the preceding example to make it read as follows:

```
# DEBUG_OPTIONS = -g # For best source-level debugging
```

That change disables debugging.

2. Remove the comment symbol from the third line in the example to add optimizations that are not needed in debugging but are useful in building release versions of applications:

```
DEBUG_OPTIONS = -XO -Xunroll=1 -Xtest-at-bottom -Xinline=5
```

When you have made these changes, you can run your makefile to build a release version of your program.

Setting Debugger Preferences

When you have configured your source directories, you're ready to perform the next step that is required in preparation for a debugging session: setting your debugger preferences and specifying a script for the debugger to use when it boots up your M2 development (dev) card.

A debugger script is a prewritten script, supplied with your M2 dev card, that the debugger follows when it boots up your card. When 3DODebug starts, it searches for a file called *3DODebug.Prefs* in the same directory that 3DODebug occupies. If there is such a file, 3DODebug loads preference information from it.

When you start 3DODebug, the Target Setup dialog opens if 3DODebug can't find a *3DODebug.Prefs* file present or if you hold down the Command key while the 3DODebug program launches.

If you want to launch 3DODebug without loading the preference information in the *DODebug.Prefs*, hold down the Command key during the launch process.

Special Mode

When the Portfolio Terminal window opens and text starts scrolling, you can significantly improve the speed of the boot process by choosing Special Mode from the File menu or by pressing the Command key and the = sign at the same time (Command+ =).

A dialog box then appears telling you that 3DODebug is running in Special Mode, and 3DODebug bypasses the normal event loop. When the boot process ends, you can deactivate Special Mode by pressing the space bar or simply clicking the mouse.

To set the debugger preferences defined by your *3DODebug.Prefs* file, you use the Target Setup dialog box, shown in Figure 1-4. If you like, you can also use the Target Setup dialog box to download a flash ROM into your 3DO M2 Development card.

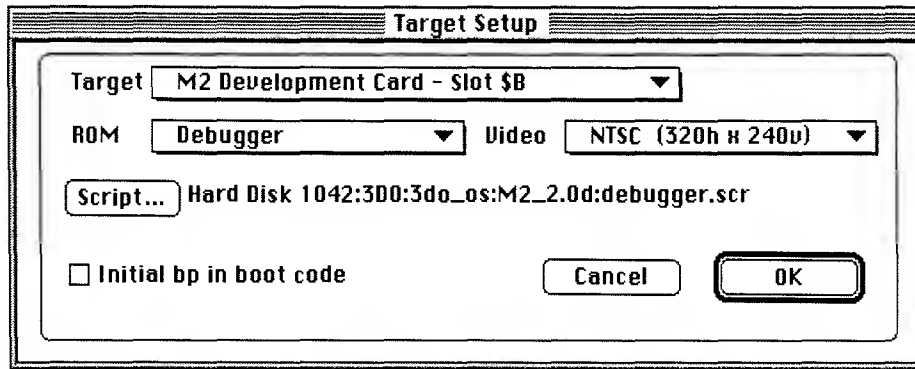


Figure 1-4 The TargetSetup dialog box.

The Target Setup dialog box contains three popup menus:

- ◆ One for selecting an M2 development card (useful when you have more than one kind of dev card installed in your Macintosh)
- ◆ One for toggling between Debugger ROM and flash ROM (if flash ROM is available).
- ◆ The video mode you are using (NTSC or PAL).

Using the Target Setup Dialog Box

To display and use the Target Setup dialog box, choose the debugger's Target | Setup menu item. Then follow these steps:

1. To configure the Target Setup dialog box for debugging, make sure that:
 - ◆ The ROM list box is set to Debugger
 - ◆ The Video list box is set to the kind of monitor you are using (NTSC 320h x240v by default)
 - ◆ The checkbox labeled Initial bp in boot code is not checked.
2. To use the debugger, you must select a debugger script by clicking the Script button. If you have debugger scripts for multiple operating systems, you can choose the script you want to use by clicking the Script button and then selecting the script for the desired OS. For this exercise, the boot script to select is:

```
<HD>:3do:3do_os:M2_2.0:debugger.scr
```

Note: If you examine the default directory structure for your M2 software, you may notice that the directory in which the debugger.scr file is stored is the same as the root directory that contains your remote folder. Under most circumstances, this is the correct setup for M2 debugging.

- Click the Target Setup dialog box's OK button. The first time you do this, your M2 developer card is booted.
- When the debugger has finished booting, the `remote>` prompt again appears in the MPW worksheet window:

```
/remote>
```

If the `remote>` prompt does not appear, press Enter to display it.

Note: Because M2 supports multitasking, additional lines of text might appear after the shell's `remote` prompt. If you press Enter or Command + Return, the text in the Portfolio Terminal window advances to the next line and another `remote>` prompt appears.

Starting the M2 Debugger

When your application's makefile is configured properly for debugging, you are ready to start the M2 debugger. To launch the debugger, follow these steps:

- Double-click the 3DO debugger icon in the `3DO:3dodebug` folder, or create an alias for the debugger icon on your desktop and double-click that. The Portfolio Terminal window then opens, as shown in Figure 1-5, and starts loading the debugger.

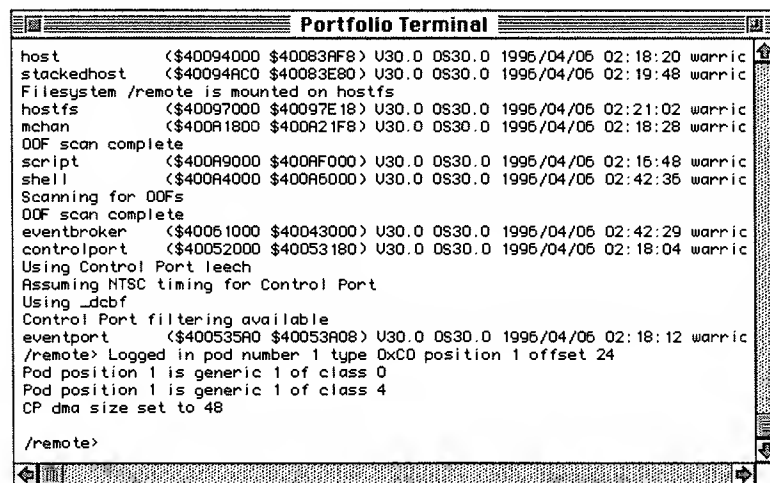


Figure 1-5 The Portfolio Terminal window.

- As the debugger loads, it displays various kinds of information messages in the Portfolio Terminal window. When the load process is complete, press the

Enter key. The debugger then displays a `remote>` prompt, as shown in Figure 1-5:

```
/remote>
```

The debugger displays this prompt because its startup working directory is

```
<HD>:3DO:3do_os:M2_2.0:remote
```

Later in this exercise, you will change this directory—along with the prompt that the debugger displays—to simplify the debugging process.

Using the Portfolio Terminal Window

When you use the M2 debugger, the Portfolio Terminal window is the center of your debugging operations. It is also your window into MPW. You can execute all MPW shell commands from the Portfolio Terminal window, in much the same way that you execute MPW commands from the Worksheet window when you are using MPW.

To open the Portfolio Terminal window manually, choose the View | Terminal menu item or type Command + T.

You can use the Portfolio Terminal window to execute MPW shell commands. The Terminal window also displays messages about the status of the 3DO M2 development card, as well as any *printf* statements executed by your program or the operating system.

To execute a 3DODebug command from the Portfolio Terminal window, you type the command and press the Enter key. You can also execute any program you have written by navigating to its directory, typing its name and any command-line arguments it requires, and pressing Enter. Then you can debug your program, if it needs debugging, by using the other features of the M2 debugger.

You can even use the Portfolio Terminal window as a help window. When you enter the command

```
help
```

in the Portfolio Terminal window, 3DODebug displays a list of shell commands that are often used in debugging (Figure 1-6 on Page 10). Learning the names of these commands is a worthwhile exercise that can save you hours of effort during your debugging sessions. And it's comforting to know that you can refresh your memory about any individual command at any time by simply executing the `help` command.

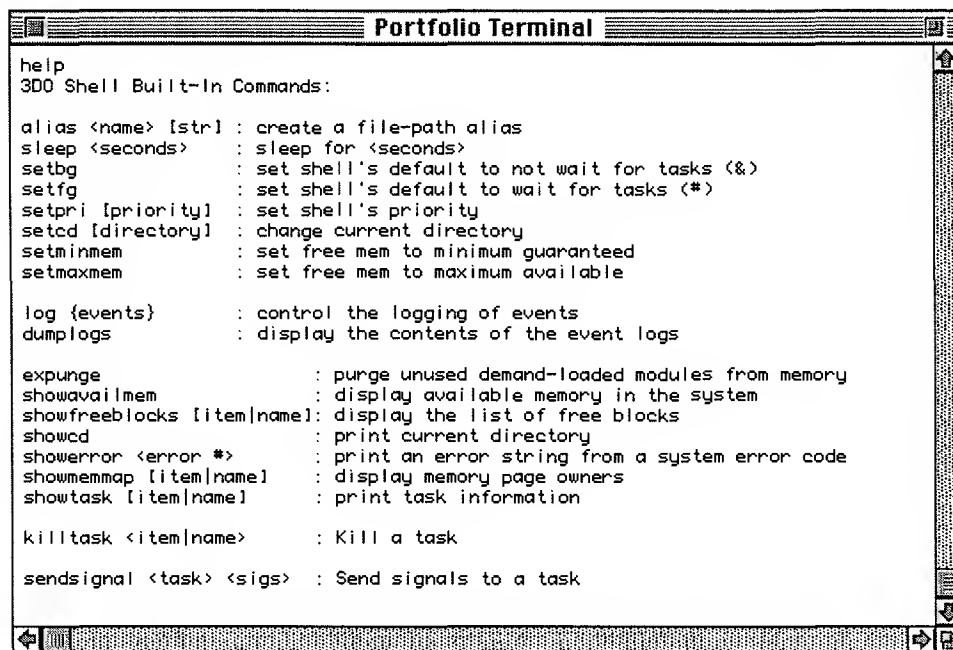


Figure 1-6 The 3DODebug help window.

Setting Up Your Source and Symbolic-Information Files

When you finish the steps outlined in the preceding section, your M2 software is set up properly to run and debug the *newview* program. Before you can start actually debugging, however, you must tell 3DODebug the locations of two kinds of files:

- ◆ Your application's source files (*newview* has only one source file, named *newview.c* file).
- ◆ Your application's symbolic-information file. The symbolic information that the M2 debugger requires is embedded in your executable, so there is no separate symbol file. So the *newview* program's symbolic-information file is the *newview* executable.

To configure your system for your application's source files and your application's executable, these are the steps to follow:

1. From the debugger's File menu, select the Directories item and then choose the Setup submenu item. In response, the debugger displays a standard Macintosh navigator dialog box like the one shown in Figure 1-7. With this

dialog box, you can set environment variables that tell the debugger where your source files and data files are.

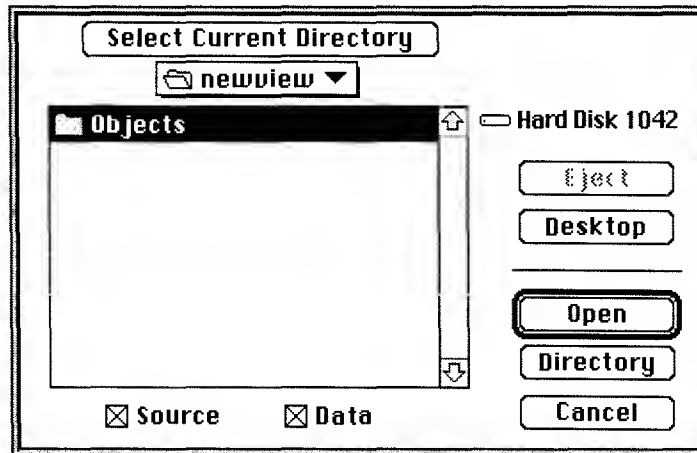


Figure 1-7 The Directories|Setup dialog box.

2. Check the Source box at the bottom of the Directories|Setup dialog box, and then use the other controls inside the dialog box to tell 3DODebug where your application's source files are.
3. Check the Source box at the bottom of the Directories|Setup dialog box, and uncheck the Data Box. Then navigate to the location of the symbolic information that is needed to debug your application. Because 3DODebug does not use separate symbolic-information files, this location is the same as the location of your application's executable.

More details about the use of the Directories|Setup dialog box are provided in the paragraphs that follow.

Note: At any time you like, you can check to see what source directory 3DODebug is currently set to access by choosing the File|Directory|Current Source Directory submenu item. Similarly, you can check to see what symbolic-information (data) directory the debugger is currently set to access by choosing the File|Directory|Current Data Directory submenu item.

Specifying File Locations with the Directories|Setup Dialog Box

When the Directories|Setup dialog box opens, notice the two checkboxes—one labeled Source and the other labeled Data—at the bottom of the dialog box. With the Source checkbox, you can specify the location of your application's source files. The Data checkbox lets you specify the location of a file that contains symbolic information for the M2 debugger.

Ordinarily, the Data (symbolic-information) directory that you specify in the Directories | Setup dialog box is the directory in which your application's executable is stored. If that's the case—and in this exercise, it is—follow these steps:

1. Check the Data button
2. Uncheck the Source button.
3. Navigate to the appropriate directory.
4. Click the button labeled Select Current Directory.

Similarly, to set an environment variable for your application's source files (in this case, just one source file, *newview.c*), you check the Source box, navigate to the folder in which *newview.c* is stored, and select that directory.

Warning: *The Data checkbox in the Directories Setup dialog box does not set up a location for data files such as SDF(.csf) files, UTF files, or other kinds of external data files that source files rely on. Instead, it is used to set the location of your application's symbolic information file—which, in Version 2.0 of M2, is your application's executable. To learn how to debug applications using external data files such as SDF and UTF files, see "Executing the debug Command" on page 16.*

By default, the *newview* makefile that is shipped with M2 reads source files that are in one directory and generates an executable that is stored in a different directory. If you use the supplied makefile, the *newview* program's source files are stored in the following directory:

```
<HD>:3DO/Examples/M2.0/Graphics/Frame/newview
```

The makefile supplied with the *newview* application places the program's executable in this same subfolder. It also places a copy of the program's executable in a different location, which is:

```
<HD>:3do:3do_os:M2_2.0:remote:Examples:Graphics:Frame:newview:
```

Note: *If you create a makefile for the newview program by executing CreateM2Make, the executable is placed in the remote folder, so you should adjust the working directory for your executable accordingly.*

Specifying the Location of Your Application's Source Files

To set a working directory for the program's source files, check the Source checkbox in the Directories | Setup dialog box but leave the Data box unchecked. Then navigate to the directory in which the *newview* program's source files are stored, which is:

```
<HD>:3DO/Examples/M2.0/Graphics/Frame/newview
```


When this pathname appears in the list box beneath the button labeled Select Current Directory, click the Select Current Directory button to set 3DODebug's source directory.

Specifying the Location of Your Application's Symbolic-Information File

To specify the location of your application's data (symbolic information) file, open the Directories | Setup dialog box again. But this time, check only the Data box in the Directories | Setup dialog box. When you have done that, navigate to the directory in which the *newview* executable is stored:

```
<HD>:3do:3do_os:M2_2.0:remote:Examples:Graphics:Frame:newview:
```

and close the Directories | Setup dialog box by clicking the Select Current Directory button.

When you finish this step, your source and data (symbolic information) directories are set up properly for source-level debugging of the *newview* program. Later, when you finish this debugging session, 3DODebug will save your pathnames in the *3DODebug.Prefs* file so you don't have to select them again when you start a new debugging session. 3DODebug will use your settings until they are changed or until the current *3DODebug.Prefs* file is discarded.

Note: You can also use the FileDirectory hierarchical menu simply to check the current working directories that 3DODebug is using to find source files and data files. To view the current working directory for source files, select the FileDirectory hierarchical menu and choose the Current Source submenu item. To view the current working directory for your application's data (symbolic information) files, choose the Current Data submenu item. In either case, 3DODebug then displays a read-only message box containing the information you are requesting.

Starting a Debugging Session

When you have started the M2 debugger and have provided 3DODebug with the locations of your application's source files and its executable file, you are almost ready to start a debugging session.

The command that you execute to debug an M2 application is the *debug* command. To invoke the *debug* command, as you'll see momentarily, you enter a line in the Portfolio Terminal window that looks something like this:

```
/remote> debug newview texcow.csf
```

Later in this section, you'll get an opportunity to start a debugging session by executing the *debug* command. First, though, there's one feature of the *debug* command that we haven't yet covered—specifically, the way the *debug* command handles data files such as the *texcow.csf* file whose name appears in the preceding example.

In M2, a file with the extension *.csf* file is an SDF (scene description format) file, which is used to create 3D scenes. Many M2 programs rely on data stored in SDF files, as well as in UTF (unified texel format) files and various kinds of audio and music files.

When you debug an application that relies on any of these kinds of files—as most M2 applications do—must take steps to ensure that 3DODebug can find those files, along with the source files used by your application.

3DODebug doesn't provide any special kinds of dialog boxes to help you specify the locations of the external data files that your source files may need to access. So, if your application uses such any such files, you must supply their locations to the M2 debugger.

One way to tell 3DODebug the location of an external data file is to specify the location of the file when you invoke the `debug` command. As you will see momentarily, you can do that in much the same way that you specify filenames on command lines when you compile and link source files using MPW.

Another way to supply the location of an external data file to 3DODebug is simply to place a copy of the file inside the working directory from which the `debug` command is issued. We'll discuss both these methods under the headings that follow.

Setting Up the Debugger for the *newview* Program

As you have seen in earlier sections, when you build the *newview* program using the default makefile provided on your distribution CD-ROM, the program's executable—named simply *newview.c*—is copied into the following directory:

```
<HD>:3do:3do_os:M2_2.0:remote:Examples:Graphics:Frame:newview:
```

However, the *texcow.csf* file comes installed in a different directory on your distribution CD-ROM:

```
<HD>:3do:3do_os:M2_2.0:remote:datafiles
```

But in the current version of the debugger, you do have to navigate to the directory in which your application's executable is stored before you can execute the `debug` command. That's because the `debug` command must always be issued from the directory in which the application being executed resides.

So, before you start debugging the *newview* application, you should use the `cd` command to navigate to the directory in which your application's executable is stored. If you like, you can navigate to the correct folder by executing a single `cd` command from the `remote>` prompt, as follows:

```
/remote> cd examples/graphics/frame/newview
```

After you execute this command, the prompt shown in the Portfolio Terminal window reads as follows:

```
remote/examples/graphics/frame/newview>
```

When your 3DODebug prompt looks like this example, you are ready to start debugging the *newview* program.

Two Ways to Supply the Location of a File

Once you are in the directory shown in the previous example, you can execute the *debug* command to start debugging the *newview* application. But you must write your *debug* command, in such a way that it tells the MPW shell where the required *texcow.csf* file is.

If you run the Macintosh File | Find utility to locate the *texcow.csf* file, you'll discover that it's in a directory named *:3DO:3do_os:M2_2.0:remote:datafiles*. One way to inform 3DODebug of the location of *texcow.csf* is to use standard UNIX-style pathname symbols to specify the pathname of the *texcow.csf* file, as follows:

```
debug newview ../../../../datafiles/texcow.csf
```

Note: If you create a *newview* makefile by executing *CreateM2Make*, the program's executable is placed in the *remote* directory, so your *cd* command should read:

```
debug newview datafiles/texcow.csf
```

Arguments to the *debug* Command

In the preceding example, the *debug* command is called with two arguments. The first argument is the word *newview*, tells 3DODebug that you want to debug the *newview* executable. The second argument is the pathname

```
../../../../datafiles/texcow.csf
```

This argument tells the MPW shell two things. It informs MPW that the *newview* application relies on an SDF file named *texcow.csf*, and it supplies 3DODebug with the pathname of the *texcow.csf* file.

The command works, but it's not terribly developer-friendly unless you really enjoy typing.

An Easier Way

An easier way to provide 3DODebug with the name and location of the *texcow.csf* file is simply to drag a *copy* of the *texcow.csf* file into the working directory from which the *debug* command is invoked: namely, the *:remote:datafiles* folder. Then you can build the *newview* program by executing the following *debug* command:

```
/remote> debug newview texcow.csf
```

When you have copied the *texcow.csf* file into the location that makes the preceding `debug` command work, you can simply execute the command. Then you can start debugging the *newview* program.

Note: *You now know how to tell 3DODebug where your application's executable file is, and where its auxiliary data files are. But how does the debugger know where your program's source files are? Remember the Directories/Setup dialog box? You use that. If you need a memory refresh, see "Specifying File Locations with the Directories/Setup Dialog Box" on page 11.*

Executing the *debug* Command

When you have finished all the pre-debugging steps described so far in this chapter, the debugger fires up and the Terminal Portfolio window displays a series of messages showing you that a debugging session has begun. These messages tell you what libraries are being loaded and provide various other kinds of startup information. The last line displayed in the Terminal Portfolio window may look something like this:

```
3DODebug: Setting breakpoint at $401eb138
```

If a line similar to this one appears in the Terminal Portfolio window, you can be fairly sure that your debugging session has started without any problems. A "setting breakpoints" message indicates that 3DODebug has started your application and has halted it at a program breakpoint—probably at a breakpoint that has been inserted automatically at the beginning of your program's `main()` function.

A Possible Problem and How to Solve It

If you don't see a "setting breakpoints" message when 3DODebug finishes loading your application, the most likely reason is that an initial breakpoint has not been set. To set one, follow these steps:

1. Select the debugger's Execution menu and check to see whether a check mark appears in front of the "Initial breakpoint in task" item, as shown in Figure 1-8 on Page 17. If there is no check mark there, select the "Initial breakpoint in task" item to set one. Then, if a check mark also appears in front of either of the "Ignore" menu items, remove it.

Execution	Target	Tools
Go		⌘G
Stop		⌘.
Kill Stopped Task		
Step Over		⌘,
Step In		⌘;
Ignore DebugBreakpoint		
Ignore breakpoints		
✓ Initial breakpoint in task		

Figure 1-8 Checking for a breakpoint at `main()`

2. Type

```
killtask newview
```

in the Portfolio Terminal window.

3. Execute the debug command again, using the format shown under the previous heading, "Two Ways to Supply the Location of a File" on page 15. You should then see the "setting breakpoints" message and the flashing screen that mark the start of a debugging session.

Checking the Status Window

To confirm that the *newview* program has been stopped at the beginning of its `main()` function, choose the View | Status menu item to open the debugger's Status window (Figure 1-9). The status window is always open unless the user explicitly closes it.

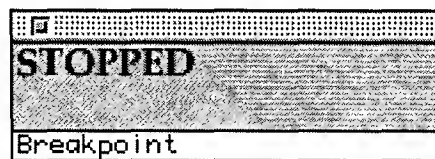


Figure 1-9 The Status window.

The Status window shows you that the *newview* program has been halted, and it shows that what stopped the program was a breakpoint. But it doesn't show where that breakpoint is. To determine the location of the breakpoint that has stopped the program, you use the Debugger's Source window.

Using the Source Window

The Source window (Figure 1-10) displays the source code of the program you are debugging. The Source window:

- ◆ Indicates the source line that represents the current location of the Power PC program counter.
- ◆ Lets you step through your source code using the Step In and Step Over commands (for details, see "Stepping Through Code" on page 29).
- ◆ Provides an Examine button that lets you scroll the text displayed in the Source window to any desired location. (To use the examine button, you type a function name, an address, or a register name in the Examine text field and then click the Examine button.)

Note: You can examine source code using the Source window only if you have compiled with the *-g* option, and you have set up directories as described in the document entitled "Getting Started."—or if you have an *.spt* file (see "Specifying Source Directories with an *.spt* File" on page 22).

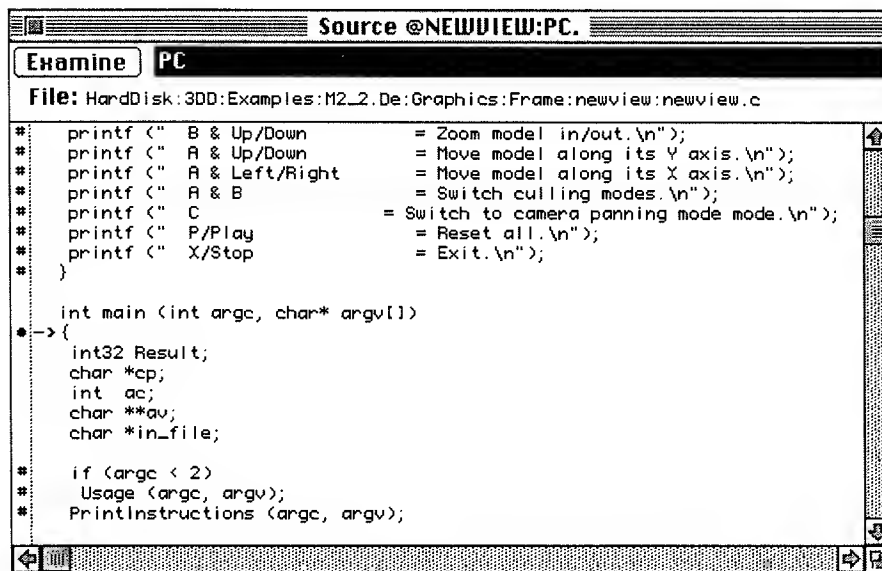


Figure 1-10 The Source window.

Opening the Source Window

To use the Source window, choose the Source menu item from the View menu or press Command + K. 3DODebug then opens the Source window, as shown in Figure 1-10.

The Source Window's PC Counter

When the Source window opens, the point where the debugger has stopped is indicated by a symbol representing the CPU's program counter (->). In Figure 1-10, the PC symbol appears at the beginning of the newview program's `main()` function.

The Source window also displays a dot (a red dot if you have a color Macintosh screen) at the beginning of each statement where a breakpoint has been set. In Figure 1-10, a breakpoint dot appears alongside the PC symbol at the beginning of the `main()` function.

Another symbol you can see in the Source window is a hash mark (#) that identifies the beginning of each source line. Hashmarks aren't just handsome; they're useful, too.

You can set a breakpoint by clicking any hash mark that is displayed in the Source window. To clear a breakpoint, you simply click its red dot that marks the breakpoint. The red dot then disappears, and the debugger doesn't automatically stop there any more.

The Disassembly Window

The M2 debugger has a Disassembly window that you can open by choosing the Disassembly item from the View menu or pressing Command + J. The Disassembly window lets you display and edit disassembled M2 object code. The debugger's Disassembly window is shown in Figure 1-11 on Page 20.

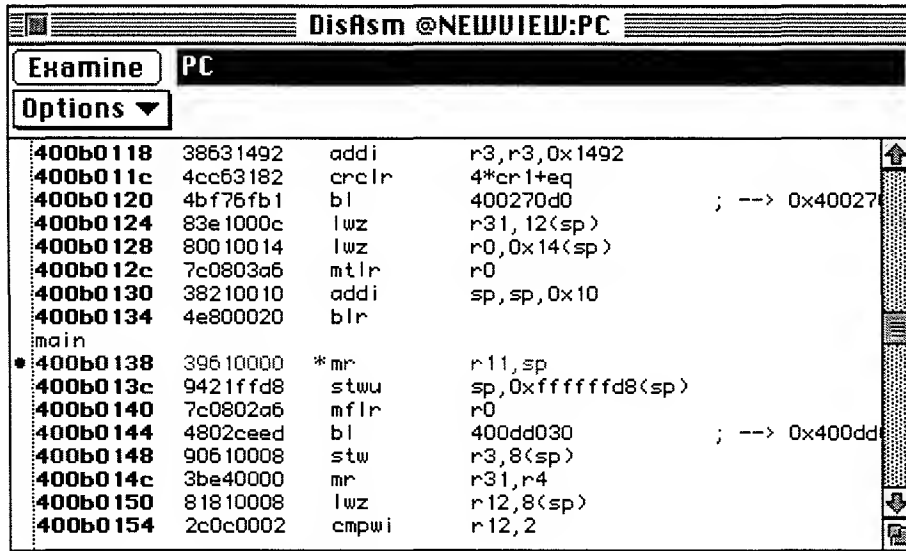


Figure 1-11 The Disassembly window.

Features of the Disassembly Window

The Disassembly window shows an assembly-language listing of your application. In the Disassembly window, an asterisk (*) in the column preceding the opcode field marks the current location of the program counter.

In the Disassembly window, just as in the Source window, breakpoints appear as red dots. In fact, on a color Macintosh monitor, the entire line in which each breakpoint appears is displayed in red in the Disassembly window.

When the Disassembly window opens, you can set a breakpoint in front of any line of code that contains an address. To set a breakpoint, just click the mouse in the far left column of the display, in front of any address. To clear a breakpoint, click on its red dot.

You can advance the text in the Disassembly window to any function by typing the name or the address of the function in the Examine text box and clicking the Examine button. To restart the debugger beginning at a stopped breakpoint, you can choose the Go command from the Execution menu or press Command + G.

Note: You can enter function names, register names, and addresses—in the Examine text boxes that appear in debugger windows.

Working with the Disassembly window

Table 1-1 lists all major operations you can perform in the Disassembly window.

Table 1-1 *Working with the Disassembly window.*

To . . .	Do this . . .
Open the window	From the View menu, select Disassembly, or press Command-J.
Set breakpoints	Click in the far left column at the point you want to set a breakpoint (see Figure 1-11).
Clear breakpoints	Click the breakpoint symbol (•) for the breakpoint you want to remove.
Set the program counter (PC)	While the program is not executing, hold down the Shift key and click to the left of the dotted line. The Debugger marks the PC address with a red asterisk (*) and displays the new PC address in the Registers window.

The Disassembly Window's Options Menu

Notice the Options popup menu just below the Disassembly window's Examine button. From the options popup menu, you can choose any of the following menu items:

- ◆ *Auto Update*: Instructs 3DODebug to update the display in the Disassembly window automatically every time an event takes place that makes a significant change in the window's display. For example, if Auto Update is set—that is, if no check appears next to the Auto Update menu item—the Disassembly window auto-updates each time the program stops at a breakpoint and every time the content of a register changes. When Auto Update is not set, the Disassembly window “frozen”—which means that it does not update automatically. Information shown in a frozen Disassembly window can be useful for later reference in some situations.
- ◆ *Hide ASCII*: Hides any ASCII text displayed in the window
- ◆ *Address*: Sets the Disassembly window to show the address of each instruction (expressed as a 32-bit hexadecimal value) in the first column of its display
- ◆ *Offset*: Uses the first column in the window to show the offset of each instruction (expressed as a 32-bit hexadecimal value), beginning at the start of the function in which the instruction appears.

Specifying Source Directories with an *.spt* File

Earlier in this chapter, in the section headed "Setting Up Your Source and Symbolic-Information Files" on page 10, you learned one method for supplying the M2 debugger with the locations of your application's source files before you begin a debugging session. Under the heading "Specifying the Location of Your Application's Source Files" on page 12, you saw how you can perform this task using the Directories | Setup dialog box.

In this section, you'll learn about another technique for accomplishing the same purpose. This alternative method for telling 3DODebug where your application's source files are is to create an *.spt* file.

The main advantage of using an *.spt* file is that it can provide 3DODebug with information about multiple source files stored in different directories. In contrast, with the Directories | Setup dialog box, you can provide 3DODebug with the location of only one source-file directory.

A simple *.spt* file is shown in Figure 1-12.

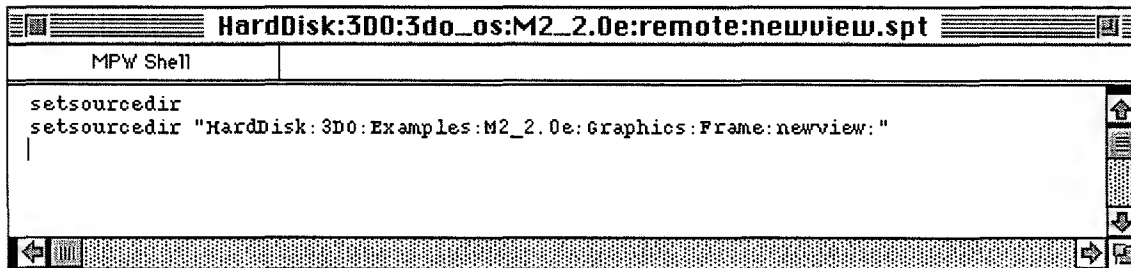


Figure 1-12 An *.spt* file.

What's an *.spt* File?

An *.spt* file is a text file that provides 3DODebug with the pathnames of the source files used by an application. When you have created an *.spt* file, you can place it in the same directory in which you have stored your application's executable. Then, when you debug an application, 3DODebug retrieves the location of your application's source files from the directories you have specified in the *.spt* file instead of referring to the Directories | Setup dialog box for the location of the application's source file.

Architecture of an .spt File

In the example of an .spt file shown in Figure 1-12, the first line—which contains just the word `setsourcedir`—clears the 3DODebug environment variable that currently specifies the current source directory, if such a variable has been set. The second line in the file set new values for the locations of the debugger's current source and data directories.

To specify the location of an executable (data) file in an .spt file, you place a line containing the word `setdatadir` in the .spt file, followed by a line that specifies the pathname of your application's executable. If you do that, however, you must still use the Directories | Setup dialog box to set the location of your application's executable.

Warning: *Because of the way in which the debugger parses spaces, pathnames that contain spaces should be enclosed in quotation marks.*

Where to Put an .spt File

If you decide to use an .spt file to debug an application, you must place it in the same directory that holds your application's executable (unless the executable is pointed to by an alias, in which case your .spt file should be in the same directory as your application's alias).

Note: *The default newview makefile that is provided on the M2 distribution CD-ROM automatically creates an .spt file. The CreateM2Make command creates one, too.*

Summary

This chapter introduced the 3DO M2 debugger and showed how to set it up for a debugging session. The setup procedures that were described in this chapter can be summed up in the following steps:

1. To start your setup procedure, open the debugger's Portfolio Terminal window by double-clicking the 3DODebug icon. (To simplify debugging, you can place an alias for that icon on your Macintosh desktop. If you decide not to do that, you can find the debugger's icon in your `:3DO:3dodebug` folder.)
2. When the debugger starts, its default working directory is:

```
<HD>:3DO:3do_os:M2_2.0:remote
```

But it is usually necessary change that working directory to one holding an M2 executable that has been compiled and linked with debugging enabled. In this chapter, the directory you switched to was the one in which *newview* sample program is stored. To make that change in your working directory, you executed the following command:

```
/remote> cd examples/graphics/frame/newview
```

3. Next, to simplify the debugging process, you used the Macintosh desktop's drag-and-drop feature to drag the *texcow.csf* file—an SDF file that *newview* relies upon—into the same working directory that you had set in Step 2.
4. When all the preceding steps were finished, you started debugging the *newview* program by entering the following debug command.

```
debug newview texcow.csf
```
5. When the *newview* application started, it was halted by a breakpoint at the beginning of its `main()` function. You opened the Source window by choosing View | Source, and the Source window showed that the *newview* program was indeed stopped at the beginning of its `main()` function, as shown in Figure 1-13.

That step completed the exercises you performed in this chapter. In Chapter 2, "Using the Debugger," you'll have a chance to complete a number of exercises that show how to use the M2 debugger to debug an application.

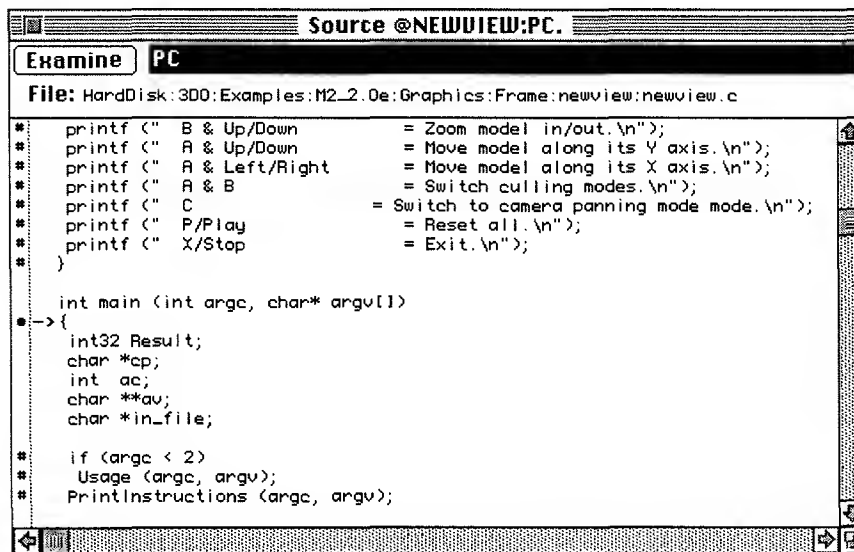


Figure 1-13 The Source window.

Using the Debugger

In Chapter 1, "Introducing the 3DO M2 Debugger," you learned how to set up the M2 debugger for a debugging session. In this chapter, you'll learn to use 3DODebug to debug an application.

The major topics covered in this chapter are:

Topic	Page
Resuming a Debugging Session	26
Using BreakPoints	27
Stepping Through Code	27
Using the BreakPoints Window	29
Working with Variables	32
Examining Data	40
Using the Task Window	43
Viewing and Modifying the Power PC Registers	43
The Stack Crawl Window	47
The Triangle Engine Disassembly Window	48
Summary	50

Resuming a Debugging Session

To demonstrate how the M2 debugger works, this chapter uses the *newview* application—the same sample program you experimented with in Chapter 1, “Introducing the 3DO M2 Debugger.”

Recall that the *newview* application has one source file, named *newview.c*. You can find that file in the `:3DO:Examples:Graphics:Frame:newview` folder on your M2 distribution CD-ROM. The program uses data supplied by a file named *texcow.csf*, which resides in a directory named `:3DO:3do_os:M2_2.0:remote:datafiles`.

In Chapter 1, “Introducing the 3DO M2 Debugger,” you learned how to set up the M2 debugger to start debugging the *newview* application. If you have not shut down the debugger since you finished Chapter 1, you can continue with the same *newview* debugging session that you began in that chapter.

If you have turned off the debugger, you'll have to restart it before you resume your debugging session and start working through the exercises performed in this chapter. You can start the debugger by double-clicking on the icon of the application you want to debug—in this case, the *newview* application.

If nobody has touched your M2 debugger since you completed Chapter 1, your debugger settings are stored in your *3DODebug.Prefs* file and 3DODebug will remember them and use them when you start the debugger up again. But you may need to refresh your memory about the steps that are required to set up the debugger's environment for a new or resumed debugging session.

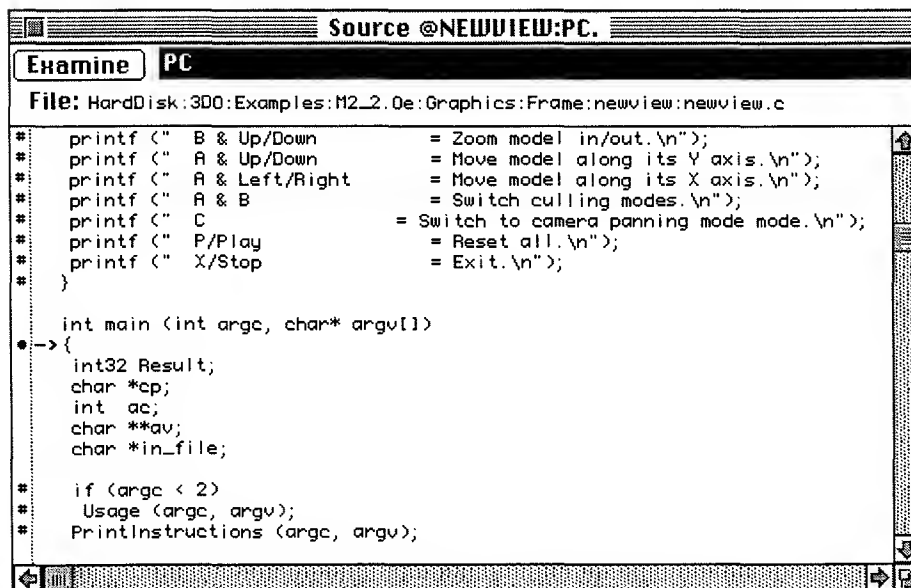


Figure 2-1 Resuming a debugging session.

If that is the case, turn back to Chapter 1's "Summary" section on Page 23 and do what's required to resume the debugging session that you started in Chapter 1.

Using Breakpoints

When you have completed the steps listed in Chapter 1's "Summary" section, the debugger's Portfolio Terminal window is open and its display looks like the one shown in Figure 2-1.

If that's the display you see now, you're at the same point in your debugging session that you were when you finished Chapter 1. Now, to more about how breakpoints work in an M2 debugging session, follow these additional steps:

1. Find the button labeled Examine at the top of the Source window. In the text box next to that button, type the function name `RotateScene`.
2. Click the Examine button. In response, 3DODebug advances the text in the Source window to a function named `RotateScene()`, as shown in Figure 2-2.

Warning: The debugger's Examine feature uses C-language case rules, so it is case-sensitive. So be sure to use the proper capitalization when you type the name of the `RotateScene` function in the Source window.

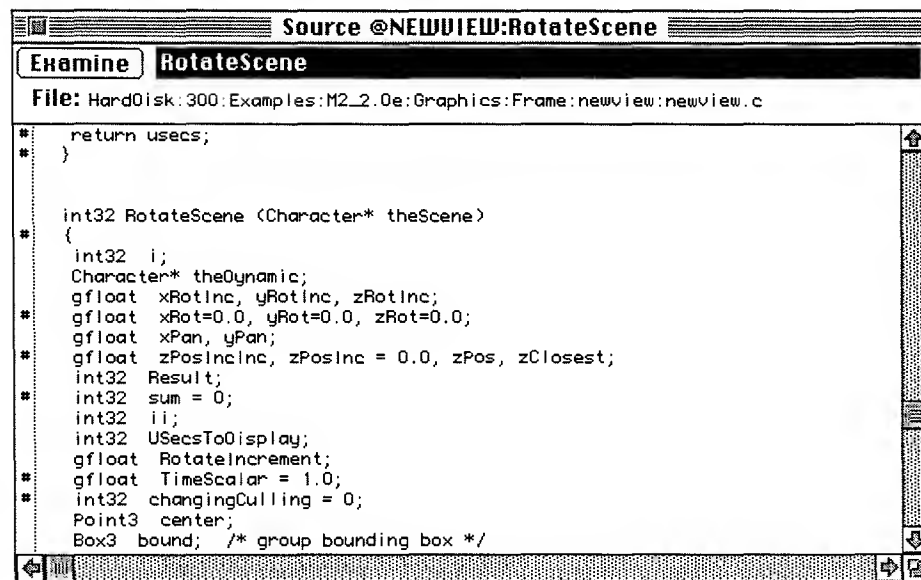


Figure 2-2 Advancing to the `RotateScene()` function.

- When the Source window display looks like the one shown Figure 2-2, scroll down to the beginning of a *while* loop that begins with the line *while (1)*, and set a breakpoint at the *cam_state = 0* statement that appears at the beginning of the *while (1)* loop (Figure 2-3). To set this breakpoint, simply click the mouse over the hash mark that appears in front of the *cam_state = 0* statement.
- Press Command + G or choose the Execution | Go menu item. The Source window's program-counter indicator (->) then advances to the beginning of the *cam_state = 0* statement, as shown in Figure 2-3.

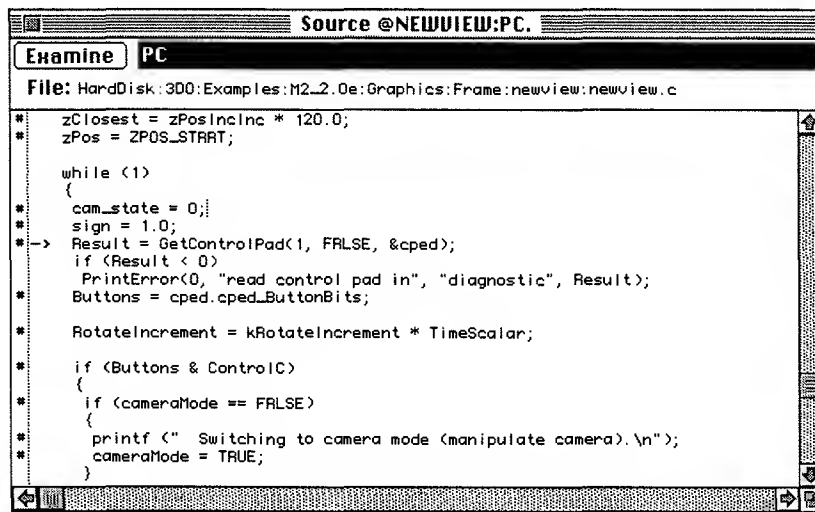


Figure 2-3 Setting a breakpoint.

- From the debugger's View menu, choose the BreakPoints command. 3D0Debug then displays the BreakPoints window shown in Figure 2-4. When the BreakPoints window opens, notice that it contains the addresses of two breakpoints: the one that the debugger set automatically at the beginning of the *newview* program's *main()* function, and the breakpoint you just set inside the *RotateScene()* function in Step 3.

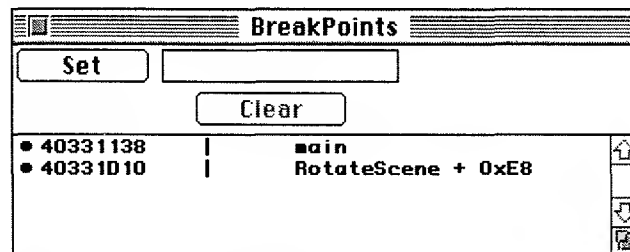


Figure 2-4 The BreakPoints window.

6. When the BreakPoints window opens, experiment with it. During a debugging session, you can use the BreakPoints window to set and clear breakpoints as well as to check their location. To set a breakpoint at the beginning of a function, you can type either its name or its address in the Set text box and then click the Set button. To set a breakpoint at an instruction that is not the beginning of a function, type the address of the instruction in the Set text box and click the Set button. To clear a breakpoint, select any breakpoint that is displayed in the window's client area and click the Clear button.

Note: You can enter function names, register names, and addresses in the Examine text boxes that appear in debugger windows..

Using the BreakPoints Window

The BreakPoints window (Figure 2-5 on Page 30) lists all currently defined software breakpoints. To open the BreakPoints window, press Command + U or select the BreakPoints command from the View menu.

Table 2-1 lists operations you can perform using the BreakPoints window.

Table 2-1 Working with the breakpoints window.

To . . .	Do this . . .
Set a breakpoint	Enter the breakpoint address or a function name into the edit box to the right of the Set button, then click Set. The source line or disassembly line is marked with a bullet (•).
Remove a breakpoint	Click on a breakpoint in the list to select it, then click Clear. The bullet in the left margin of the Source or Disassembly window is replaced with a pound sign (#).

Stepping Through Code

The M2 debugger provides two kinds of step commands:

- ◆ The Step In command, which you can execute by typing Command + semicolon (;) or by choosing the Execute | Step In menu command.
- ◆ The Step Over command, which you can execute by typing Command + comma (,) or by choosing the Execute | Step Over menu command.

The Step Over command treats a subroutine call and all the code within the subroutine called as one instruction. When you execute a Step Over command at a subroutine, the debugger stops at the next instruction or source statement after

the subroutine call. If you execute a Step In command at a subroutine call, the debugger stops at the first instruction within the subroutine. If the code does not contain a subroutine call, Step Over and Step In behave identically.

When you step through a program, the topmost window determines the behavior of the debugger. If a Source window has the focus, a Step command performs a source-level step. If the topmost window is the Disassembly window, a single step advances one assembly instruction. If neither of these windows has the focus, the kind of step that is performed defaults to the kind of step that was most recently performed. Stepping initially defaults to assembly-language level.

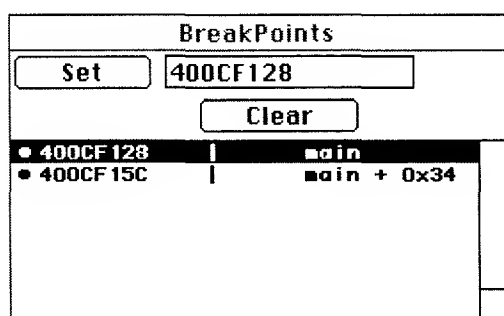


Figure 2-5 BreakPoints window

Using the Step In Command

In the previous section, “Using BreakPoints,” you set a breakpoint that stopped the *newview* application inside a function named `RotateScene()`. Now, with *newview* still halted at that point, you can perform a pair of exercises that demonstrate the use of the Step In and Step Over commands.

To see how the Step In command works, try this exercise:

1. With the Source window open, and with a breakpoint still set at the `cam_state = 0` statement shown in Figure 2-3 on Page 28, execute two Step In commands by pressing Command + semicolon (;) twice. Because the Step In command advances the program counter one instruction at a time, the PC symbol (->) is now situated in front of the `GetControlPad()` function, as shown in Figure 2-6.

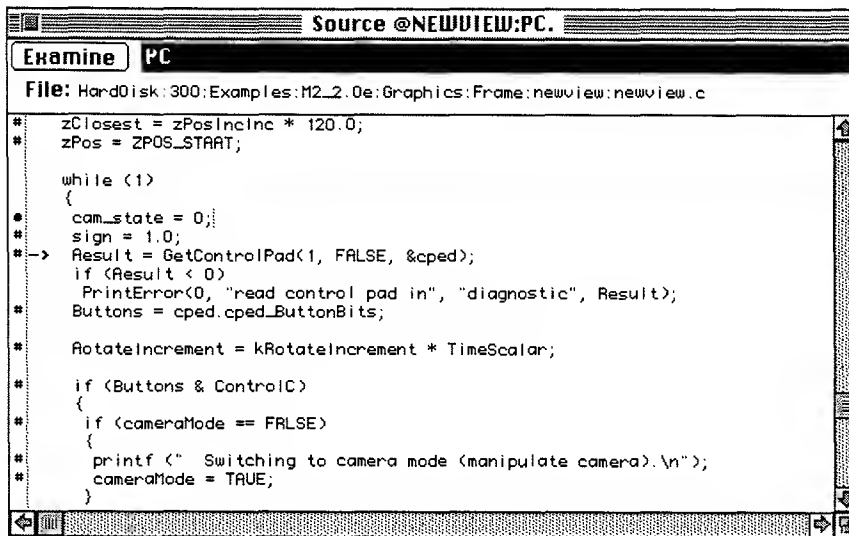


Figure 2-6 Using the Step In command.

Note: Both the Source window and the Disassembly window adjust their displays to keep the location in the examine field in the window. To update the source or disassembly window to show the current location of the program counter, set the examine field to PC. Because the Source window has its Examine field set to RotateScene, the PC only shows up if it happens to be in the current display. Set the examine field back to PC, so the window will track the PC as we step.

Using the Step Over Command

To see how the Step Over command works, follow these steps:

2. Check the Source window to make sure that a breakpoint is still set at the beginning of the `cam_state = 0` statement and that the PC indicator (`->`) is still situated at the beginning of the `GetControlPad()` function, as shown in Figure 2-6.
3. Execute three Step Over commands by pressing Command + comma (,) three times.
4. Because the Step Over command steps the program counter over each function that is encountered, the PC symbol (`->`) is now situated in front of the second `if` loop inside the `while (1)` loop, as shown in Figure 2-7 on Page 32.

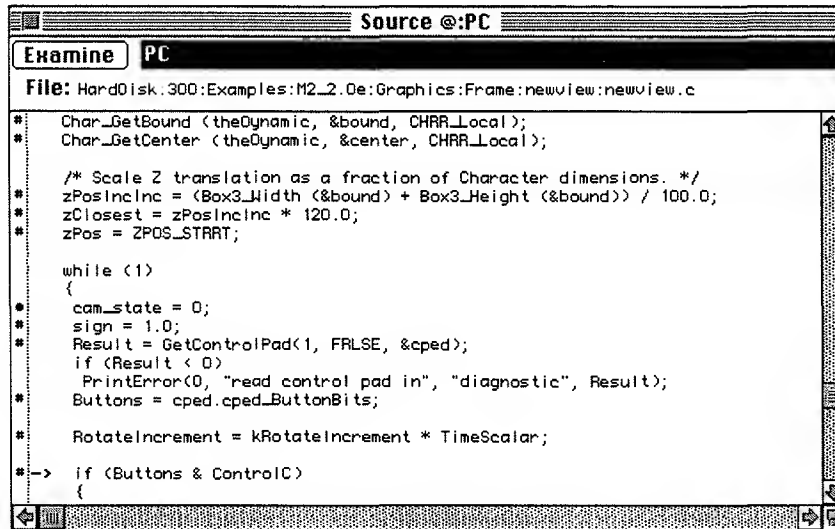


Figure 2-7 Executing the Step Over command.

Working with Variables

The 3D0 debugger provides a Variables window (Figure 2-8) that lets you look at structures, evaluate variables, and dereference pointers. You can open the Variables window by pressing Command + M or by selecting the View | Variables menu item.

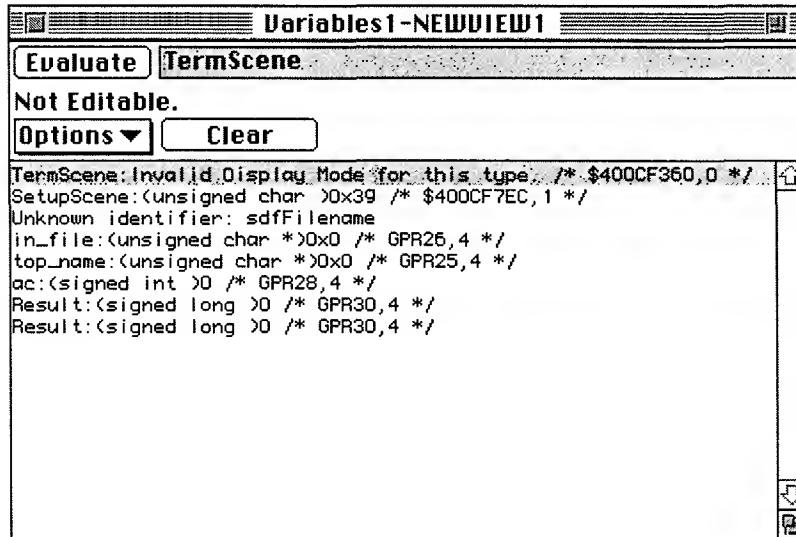


Figure 2-8 Variables window.

Features of the Variables Window

The primary job of a Variables window is to let you examine variables and data structures and to provide you with the addresses and the contents of pointers to variables. You can also use Variables window to change the values of variables, and to parse C-style expressions.

You can open as many Variables windows as you like during a debugging session. Each Variables window that you open is assigned a number as part of its name. That number appears alongside the name of the window in the window's title bar. For example, the designation of the Variables window shown in Figure 2-8 is Variables 1.

Working with the Variables Window

Table 2-2 lists and describes operations that you can perform using the Variables window.

Table 2-2 *Working with the Variables window.*

To . . .	Do this . . .
Add a variable or other expression to the display	<ul style="list-style-type: none"> ◆ Enter the variable you want to display into the text edit field at the top of the window. ◆ Press Return or click the Evaluate button. This operation places your selection in the window and displays its current value.
Examine a variable from any window	Select a variable in the Source window by double-clicking on it. Then press Command + D or choose Send Variable Data from the View menu to examine that variable in the Variables window.
Delete a variable	<ul style="list-style-type: none"> ◆ Click on the variable you want to delete. The variable name is displayed in the text field at the top of the window and the Clear button is enabled. ◆ Click on the Clear button or press the Clear key on the Macintosh keyboard (you might need to do this twice if the variable is not editable).
Change variable values interactively	<ul style="list-style-type: none"> ◆ Click on the variable. The variable name is displayed in the text field at the top of the window. When you select the variable, its current value is displayed in the Value field. ◆ Type the new value into the Value field and press Return or Enter to change the value.

Table 2-2 Working with the Variables window (Continued).

To . . .	Do this . . .
Dereference a struct	Command-double-click on the struct pointer. The structure expands to reveal its individual files. If the field cannot be dereferenced, you hear an alert signal.
Select a window for receiving data	Choose Receive Data from the Options menu of the desired Variables window. To compare the values of variables at different states in the program, you can create several Variables windows and select a different one for receiving data at any state that's of importance.

Two Ways to Use the Variables Window

When you want to view or change the contents of a variable, there are two main ways to specify the variable that you're interested in. The first way is to type the name or the address of the variable, or the name or address of a pointer to the variable, in the text box that appears at the top of the Variables window. Then you can click the Evaluate button to evaluate the variable.

The other way to let the Variables window know what variable to evaluate is to highlight the name of the variable in the Source window and then execute a command named Send Variable Data. In this section, you'll learn how to use both these methods.

This section contains three exercises. In the first exercise, you'll learn how to determine the value of a variable using the Variables window. In the second exercise, you'll get an opportunity to change the value of a variable and observe the effect that the change has on the newview *application*. In the third exercise, you'll see how the Variables window handles arrays.

Examining the Value of a Variable, Method 1

To evaluate a variable using the Evaluate button and the Evaluate text box in the Variables window, follow these steps:

1. With the *newview* application running and stopped at the breakpoint shown in Figure 2-6, type the word `cam_state` in the Evaluate text box and press the Evaluate button. The Variables window responds by displaying the value of the `cam_state` variable, as shown in Figure 2-9.

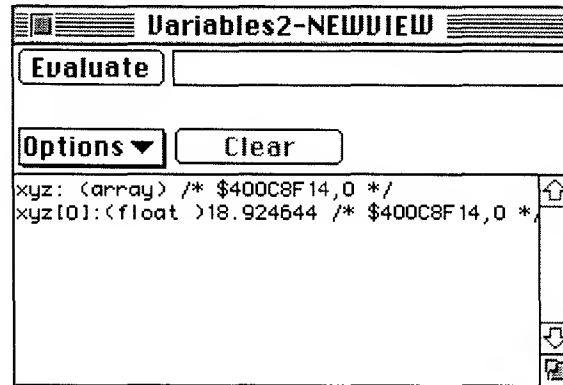


Figure 2-9 Using the Variables window.

2. When the Variables window opens, click the Options button and examine the window's Options popup menu. From the Options popup menu, you can select from the following options:
 - ◆ *Auto Update*: Automatically refreshes the window during stepping operations or when the program executing on the target hits a breakpoint. Setting Auto Update slows the general operation of the Debugger, but provides a more current and accurate display.
 - ◆ *Receive Data*: A black dot appears in front of this item when the Variables window is receiving data from the Source window. The receiving of data from the Source window is described under the next heading, "Changing the Value of a Variable."
 - ◆ *Data Types*: Select one of the items listed in this group to specify the format in which you want the Variables window to display data.

Examining the Value of a Variable, Method 2

This is the second method for examining the value of a variable using the Variables window:

1. Select the `cam_state` item that you placed in the Variables window in the previous exercise.
2. Delete the item from the display in the Variables window by clicking the Clear button.
3. Close the Variables window by clicking its Close box.
4. Highlight the `cam_state` variable in the debugger's Source window (Figure 2-10).

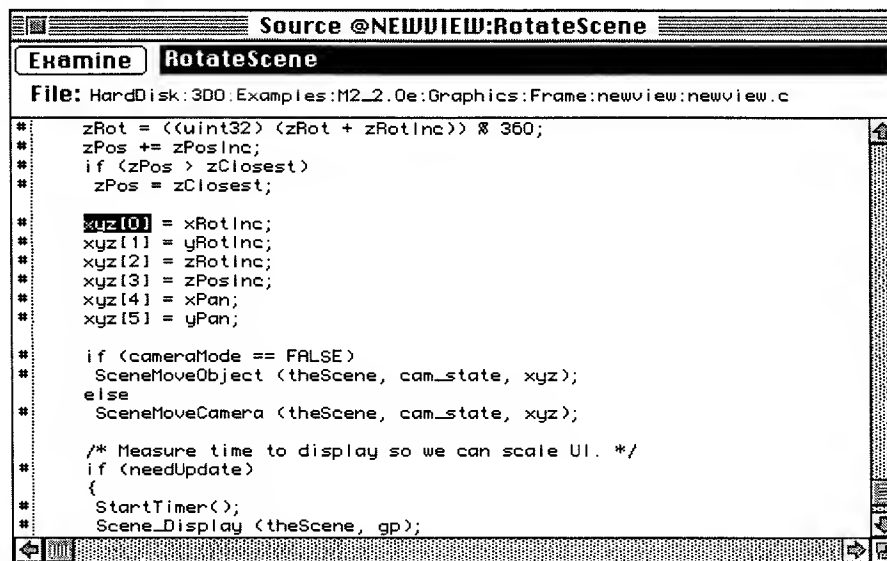


Figure 2-10 Highlighting a variable in the Source window.

5. Type Command + D or choose the Send Variable Data item from the View menu. The Variables window reappears, with the value of the `cam_state` variable once again displayed, exactly as it appeared earlier in Figure 2-9 on Page 35.

Only one Variables window can be set to receive data. You can select that window by choosing Receive Data in its Options menu. The system automatically turns off all other Variables windows.

When you send a variable from another window to the Variables window, the window receiving data comes to the front.

Note: *If there is only one Variables window, it is always the window that receives variable data.*

Changing the Value of a Variable

You can use the Variables window to modify the values of variables. Then you can see how your program executes when the values of those variables are changed. To see how this kind of operation works, follow these steps:

1. Click on `cam_state` item that you sent to the Variable window the preceding exercise. The value of the variable—0x0—then appears on a new line labeled Value (Figure 2-11).

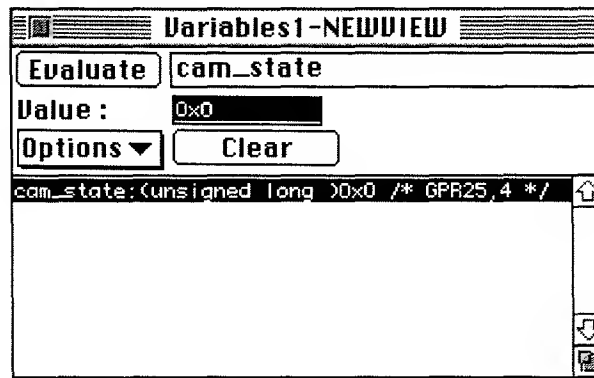


Figure 2-11 Changing the value of a variable.

2. In the Variables window's Value text box, change the value 0x0 to 0x36 and press Return. That changes the value of the `cam_state` variable from zero to 0x36.

Exercise: Seeing How It Works

To demonstrate how you can change the values of a variable, and how that can affect your application, try this exercise (note that you have to modify three different variables each time you want to observe the results of the exercise because two of the variables get reset each time through the loop):

1. Near the very end of the newview application, in the `RotateScene()` function, place a breakpoint at this line:

```
xyz[0] = xRotInc;
```
2. Allow newview to run until it stops at the breakpoint.
3. Pass the following variables to the variables window:
 - ◆ `xRotInc`
 - ◆ `yRotInc`
 - ◆ `zRotInc`
 - ◆ `needUpdate`
 - ◆ `cam_state` (type this one into the Evaluate field).
4. Change the value of `xRotInc` to 30.000000.
5. Change the value of `needUpdate` to 0x1.
6. Change the value of `cam_state` to 0x1 (or 0x3, 0x5, or 0x7).
7. Choose **Execute | Go**. The model rotates around the *x* axis.

Another Exercise

Now do this:

1. Change the value of `yRotInc` to 30.000000.
2. Change the value of `needUpdate` to 0x1.
3. Change the value of `cam_state` to 0x2 (or 0x3, 0x6 or 0x7).
4. When you choose **Execute | Go**, the model rotates around the *y* axis.

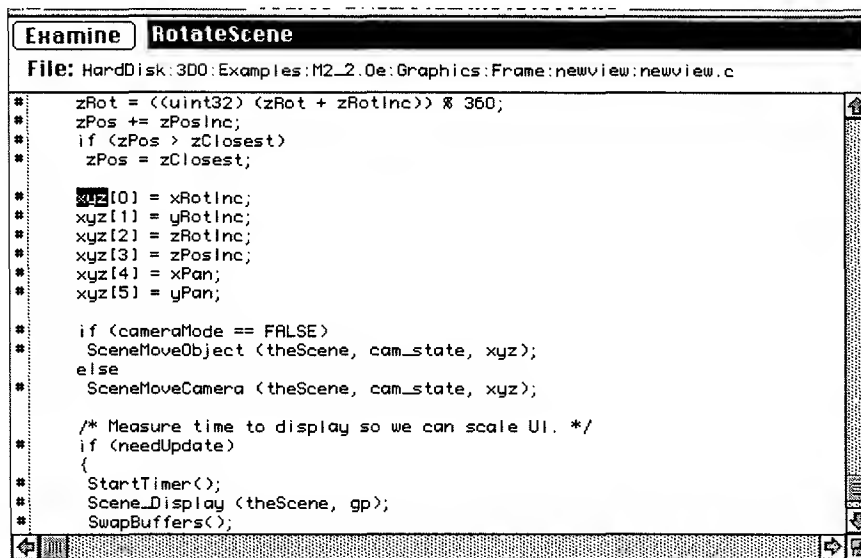
One More

Finally:

1. Change the value of `zRotInc` to 30.000000.
2. Change the value of `needUpdate` to 0x1.
3. Change the value of `cam_state` to 0x4 (or 0x5, 0x6, or 0x7).
4. When you choose **Execute | Go**, the model rotates around the *z* axis.

Working with an Array

To learn how the Variables window manages arrays, leave the `RotateScene()` function displayed in the Source window—as illustrated in Figure 2-10 on Page 36—but scroll farther down into the function's source code until you reach the definition of an array named `xyz[]`, as shown in Figure 2-12.



```

Examine RotateScene
File: HardDisk:3D0:Examples:M2_2.0e:Graphics:Frame:newview:newview.c

* zRot = ((uint32) (zRot + zRotInc)) % 360;
* zPos += zPosInc;
* if (zPos > zClosest)
*     zPos = zClosest;
*
* xyz[0] = xRotInc;
* xyz[1] = yRotInc;
* xyz[2] = zRotInc;
* xyz[3] = zPosInc;
* xyz[4] = xPan;
* xyz[5] = yPan;
*
* if (cameraMode == FALSE)
*     SceneMoveObject (theScene, cam_state, xyz);
* else
*     SceneMoveCamera (theScene, cam_state, xyz);
*
* /* Measure time to display so we can scale UI. */
* if (needUpdate)
* {
*     StartTimer();
*     Scene_Display (theScene, gp);
*     SwapBuffers();
* }

```

Figure 2-12 Working with Arrays.

Then follow these steps:

1. From the debugger's View menu, select the Variables command to display the Variables window.
2. When the Variables window appears, type the letters xyz in the Evaluate text box and then click the Evaluate button. The Variables window then displays the address of the first element in the xyz [] array, as illustrated in Figure 2-13.

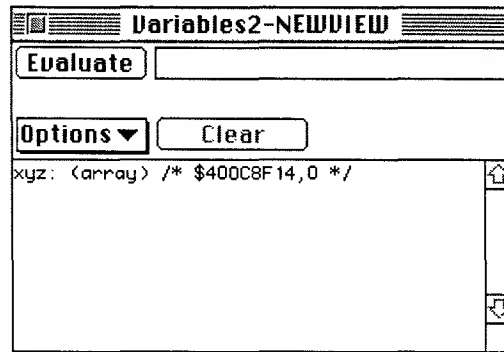


Figure 2-13 Evaluating an array.

3. To display the value of the first element in the xyz [] array, shift the focus to the Source window and highlight the characters xyz [0] in the Source window (see Figure 2-14).

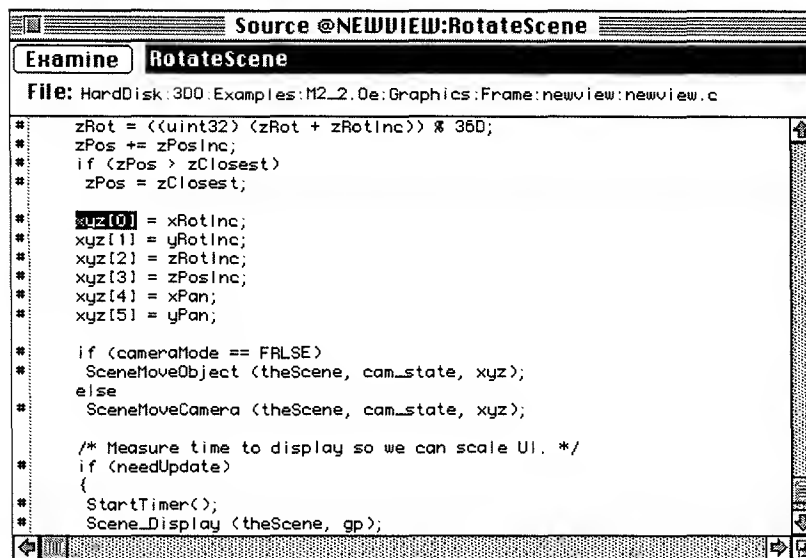


Figure 2-14 Selecting an array variable in the Source window.

4. With the `xyz[0]` variable element selected, choose the View | Send Variable Data menu item. 3DODebug then displays the value of the `xyz[0]` variable element in the Variables window (Figure 2-15). (Alternatively, you can pass `xyz` to the Variables window and then Command-double-click to expand the array. Note, however, that this technique does not work with arrays of structs.)

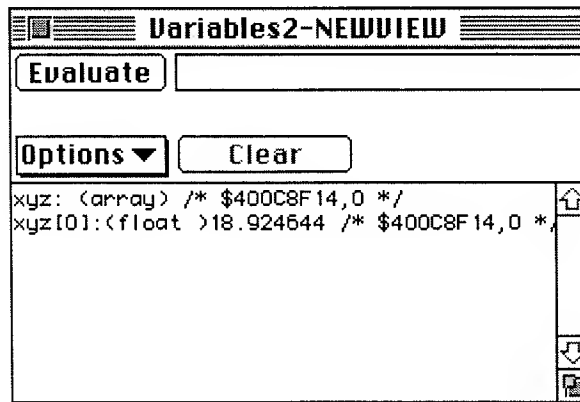


Figure 2-15 Evaluating an element in an array

Examining Data

The M2 debugger provides two windows for examining data: the Data window and the Memory Dump window. Each of these windows is examined in this section.

Using the Data Window

The debugger's Data window (Figure 2-16) lets you examine data that is used by your application. To display a block of data in the Data window, type its address in the Examine text box and click the Examine button. The data you have requested is then displayed in the window's client area.

With the Data window, you can display and edit target machine memory as bytes, half-words (16 bits), and words (32 bits). The data can be presented in hexadecimal, decimal, ASCII, or binary format.

You can open one or more data windows by choosing the View | Data menu item or by pressing Command + B.

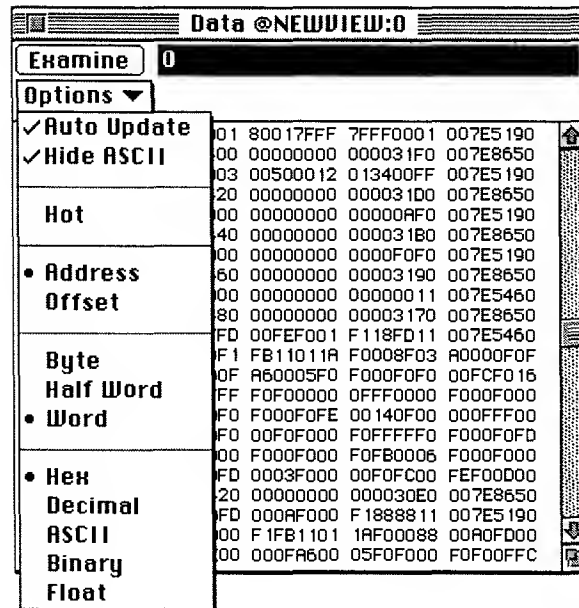


Figure 2-16 The Data window.

As Figure 2-16 illustrates, the Data window is equipped with a popup menu from which you can choose the following items:

- ◆ **Auto Update:** When Auto Update is selected (checked), and program counter is currently in the Examine field, the contents of the Data Window update to the current PC location each time the M2 is stopped, either at a breakpoint or when stepping through code. When Auto Update is not checked, the window is “frozen”—that is, it does not refresh automatically. Information displayed in a frozen Data window can be useful for later reference in some situations.
- ◆ **Hide ASCII:** When data is displayed in the Data window as text, the Hide ASCII menu item has no effect. When data is displayed in the hexadecimal, decimal, binary, or float format, the Hide ASCII item disables the ASCII representation in the far right column of the window.
- ◆ **Hot:** When this menu item is selected, changes in the Data window also appear in other windows.
- ◆ **Address:** Sets the Disassembly window to display the address of data relative to 0. Addresses are displayed as 32-bit hexadecimal values.
- ◆ **Offset:** Uses the first column in the window to show the offset of data, relative to the contents of the Examine field (PC, function name, address).
- ◆ **Byte:** Displays data as bytes. This option has no effect when you view ASCII data, since it is character based.

- ◆ *Half Word*: Displays data as 16-bit words. This option has no effect on the ASCII data display, which is character based.
- ◆ *Word*: Displays data as 32-bit long words. This option has no effect when you view ASCII data since it is character based.
- ◆ *Data types*: This group of items lets you select the format of the data that is displayed. The Float data type is similar to the Power PC 602 and Gfloat formats.

Using the Memory Dump Window

The debugger's Memory Dump window lets you dump target RAM as a binary image file onto the Macintosh. To display the Memory Dump window, choose the Memory Dump item from the Target menu.

The Memory Dump window is shown in Figure 2-17 on Page 42.

To use the Memory Dump window, follow these steps:

1. Type the starting and ending addresses of the data you want to see in the Start address and End address text boxes
2. Click the Dump button. The debugger then displays a Standard File dialog box with the default filename "3DODDebug_Dump" entered in its filename box.
3. If you don't like this name, change it. When the displayed name meets with your approval, select a directory and click OK. The debugger then creates a binary file containing the specified data.

Warning: *3DODDebug has no way of detecting the contents of the range of memory you select. And you can select any range of memory, even a block of memory that contains system code. So, avoid crashes, be sure to specify valid ranges of memory.*

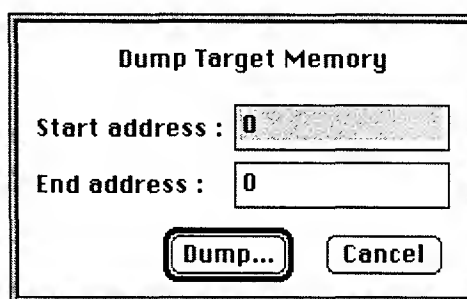


Figure 2-17 The Memory Dump window.

Using the Task Window

The Task window, shown in Figure 2-18, displays the name of the task being debugged and provides information about that task. When you execute the debug command to debug an application, the name of the task you are debugging appears in the Task window.

To open the Task window, choose the View | Task menu command.

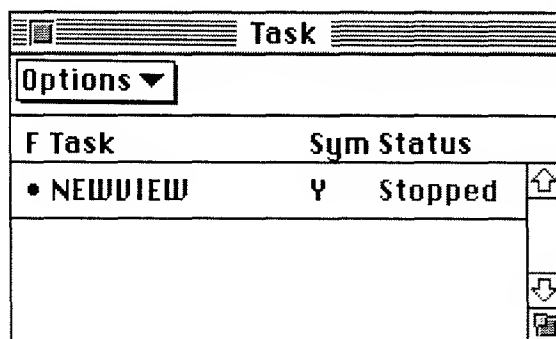


Figure 2-18 The Task window.

The fields shown in the Task window are:

- ◆ *F Task*: Displays the name of the task that is currently in focus—that is, the name of the task that is currently being debugged.
- ◆ *Sym*: Displays the letter Y (for Yes) if a symbolic information file for the specified task has been loaded, or displays the letter N (for No) if no symbols have been loaded.
- ◆ *Status*: Indicates whether the specified task is running or stopped.

The Task window is equipped with an Options popup menu, but only one item—the Remove Task item—is currently activated. When you choose the Remove Task item, 3DODebug unloads its symbols. You can choose Remove Task when you are no longer interested in debugging the specified task.

Viewing and Modifying the Power PC Registers

The M2 debugger provides three windows for viewing and modifying Power PC registers: the PPC User Registers window, the PPC FP (floating-point) Registers window, and the PPC Supervisor Registers window.

The Power PC User Registers Window

The Power PC User Registers window (Figure 2-19) displays the current contents of the Power PC's user registers. When your application is stopped, you can modify the contents of the registers shown in the PPC User Registers window by either typing in new values or clicking the mouse.

To open the Power PC User Registers window, choose the View | PPC Users Registers menu command.

PPC Registers Window

CPU HALTED, REGISTER MODIFICATION ENABLED

PC: 40215d34	LR: 40215d2c	CTR: 40026634	CR: 24000000
XER: 00000000			

r0: 40215d2c	r8: 00003030	r16: 00000000	r24: 00000000
r1: 400c8ee0	r9: 00000000	r17: 00000000	r25: 00000000
r2: 00000000	r10: 00000000	r18: 00000000	r26: 00000000
r3: 00000000	r11: 400c8ee0	r19: 00000000	r27: 400c901c
r4: 0000b030	r12: 4006d8e8	r20: 00000000	r28: 00000000
r5: 40241034	r13: 00000000	r21: 00000000	r29: 00000000
r6: 400114b0	r14: 00000000	r22: 00000000	r30: 402fac60
r7: 0000f930	r15: 00000000	r23: 00000001	r31: 4006d924

CR0	CR1	CR2	CR3	CR4	CR5	CR6	CR7
0 0 1 0 0	1 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0
L G E S F	F V O L	G E S L	G E S L	G E S L	G E S L	G E S L	G E S L
T T Q O X	E X X T	T Q O T	T Q O T	T Q O T	T Q O T	T Q O T	T Q O T

XER: 0
S O C
O U A

Byte Compare Byte Count

☐ Reserved

fmr f19, f18

Figure 2-19 The Power PC User Registers window.

To modify a single digit in a register using the mouse, click the mouse button twice and hold it down the second time. A popup menu then appears next to the register you have selected (Figure 2-20), and you can modify the contents of the register by selecting one of the entries in the popup menu. Alternatively, you can click a register once, type in a new value, and then press Enter or Return.

The bit representations of the CR and XER registers are displayed at the bottom of the window. These bit representations are not modifiable.

When the User Registers window is open, you can move from register to register either by moving the mouse or by pressing the tab key.

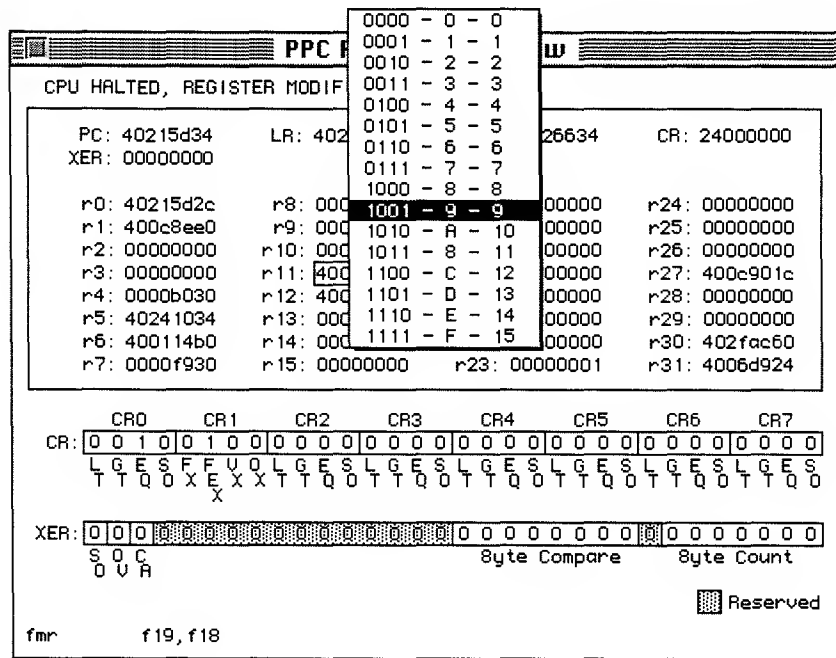


Figure 2-20 A popup menu in the Power PC User Registers window.

The Power PC FP Registers Window

The Power PC FP Registers window displays the current contents of the Power PC's floating-point registers. When your application is stopped, you can change the contents of the registers shown in the FP Registers window in the same way you modify the contents of the registers in the User Registers window—by either typing in new values or using popup menus that appear when you click the mouse.

When the FP Registers window is open, you can move from register to register either by moving the mouse or by pressing the tab key.

To open the Power PC FP Registers window, choose the View | PPC FP Registers menu command.

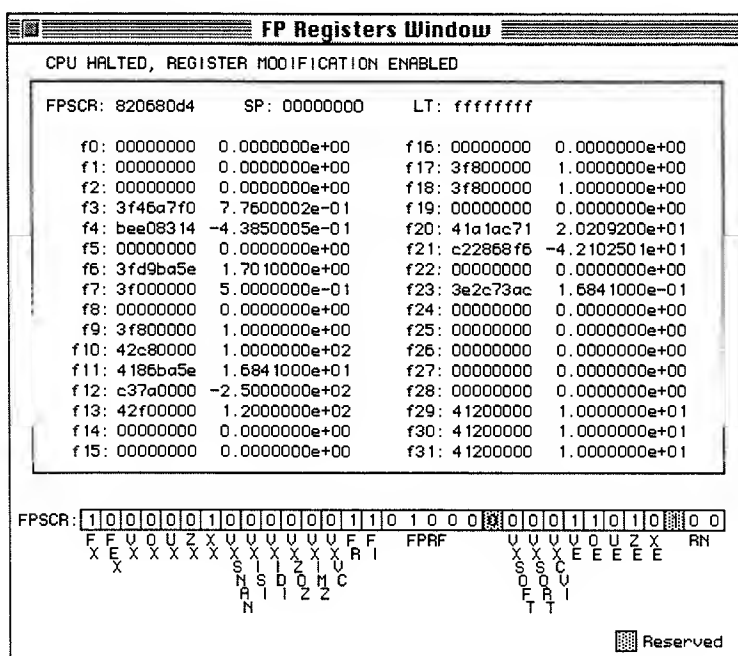


Figure 2-21 *The Power PC FP Registers window.*

The Power PC Supervisor Registers Window

The Power PC Supervisor Registers window displays the contents of the supervisor registers in the M2's Power PC microprocessor. The registers shown in Supervisor Registers window are read-only and cannot be modified.

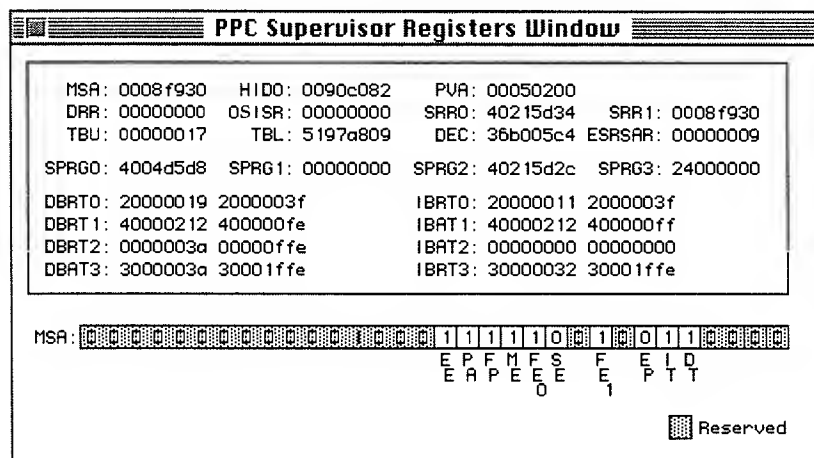


Figure 2-22 *The Power PC Supervisor Registers window.*

To open the Power PC Supervisor Registers window, choose the View | PPC Supervisor Registers menu command.

The Stack Crawl Window

The Stack Crawl window displays the calling chain—that is, the stack frames and return addresses—of the function on which the program is currently stopped.

To open the Stack Crawl window, select the View | Stack Crawl menu command.

The most recent frames appear at the bottom of the window, the oldest frames at the top. The Stack Crawl window walks the stack frames, displaying the frame address and the return address from the frame.

If a symbol name is found for the return address, the calling function is displayed with an offset from the beginning of the function where the call came from.

For example, in the “Using BreakPoints” section that starts on Page 27, you stopped the *newview* program at a breakpoint inside a function named *RotateScene()*. That breakpoint was placed at the beginning of a loop named *while (1)*, as shown in Figure 2-23.

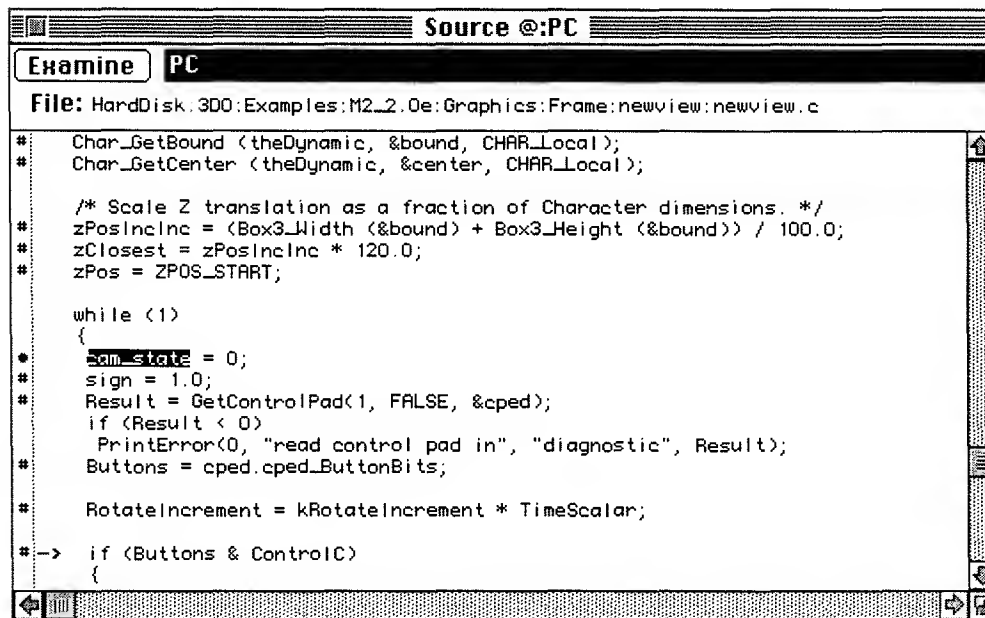


Figure 2-23 Stopping a program with a breakpoint to view the Stack Crawl window.

When an application stops at a breakpoint, the Stack Crawl window (Figure 2-24) displays the current call chain—that is, the stack frames and return addresses of the function in which a program is currently stopped. To open the Stack Crawl window, you choose the Stack Crawl command from the debugger's View menu.

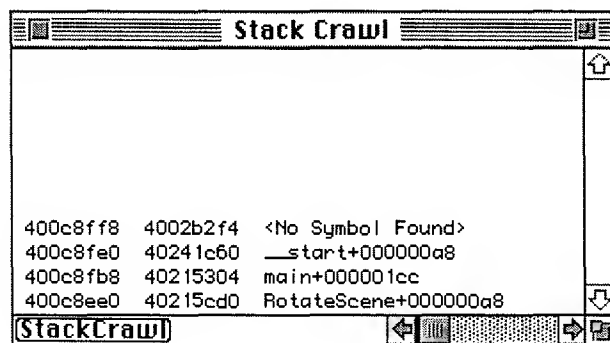


Figure 2-24 The Stack Crawl window.

When the Stack Crawl window opens, the most recently encountered stack frame appears at the bottom of the window. The second line from the bottom displays information about the second most recent stack frame, and so on. In other words, the Stack Crawl window “walks” the stack frames, displaying both the address of each frame and the return address from each frame.

If a symbol name is found for the return address of a stack frame, the name of the calling function is displayed, along with an offset from the beginning of the function from which the call came.

The Stack Crawl window shown in Figure 2-24 shows the calling sequence of the *newview* program when it stopped at the breakpoint shown earlier in Figure 2-23 on Page 47. As you can see, the `RotateScene()` function is the last function that was called. Figure 2-24 also shows that:

- ◆ The `RotateScene()` function was called from the program's `main()` function.
- ◆ The `main()` function was called from the C runtime library's `__start()` function.

The Triangle Engine Disassembly Window

The Triangle Engine Disassembly window (Figure 2-25) lets you disassemble the instructions that your application is sending to the Triangle Engine.

To open the Triangle Engine Disassembly window, choose the View | Triangle Disassembly menu command.

The range of addresses being disassembled is specified at the top of the window. To change this range, you can either change these values or enter new values.

The Triangle Disassembly window allows only the viewing of instructions—it doesn't give you any control over executing the instructions.

The Triangle Engine Disassembly window disassembles ranges of memory as if they were Triangle Engine instructions. On each line of code displayed in the Triangle Engine Disassembly window, the first column shows a memory address, and the second column shows the contents at that address. The other information shown in each line is the Triangle Engine disassembly of the code at the specified address.

Each time you close and reopen the Triangle Engine Disassembly window, the disassembly shown in the window is recalculated.

Triangle Engine Disassembly		
Disassemble From 40132008 to 40133007		
40132a1c 42920000 //	Y1 =	288.000000
40132a20 43020000 //	Red =	4.000000
40132a24 3f79db23 //	Green =	115.712006
40132a28 3f79db23 //	Blue =	115.712006
40132a2c 3f79db23 //	Alpha =	115.712006
40132a30 3f800000 //	1/w =	0.000000
40132a34 3dc534ed //	X2 =	83.307861
40132a38 4088e12c //	Y2 =	142.073242
40132a3c 402240ab //	Y3 =	548.041748
40132a40 428c0000 //	x_step0 =	192.000000
40132a44 43060000 //	x_step1 =	96.000000
40132a48 3f7f7cee //	x_longstep =	1015.808105
40132a4c 3f7f7cee //	xy_step0 =	1015.808105
40132a50 3f7f7cee //	xy_step1 =	1015.808105
40132a54 3f800000 //	xy_longstep =	0.000000
40132a58 3dc574fa //	dy_0 =	87.311035
40132a5c 40730f9e //	dy_1 =	816.976074
40132a60 3feb405e //	dy_long =	692.022949
40132a64 42900000 //	d/dx Red =	32.000000
40132a68 43080000 //	d/dx Green =	16.000000
40132a6c 3f7b22d1 //	d/dx Blue =	118.272003
40132a70 3f7b22d1 //	d/dx Alpha =	118.272003
40132a74 3f7b22d1 //	d/dx 1/w =	0.462000
40132a78 3f800000 //	slope Red =	0.000000
40132a7c 3dc55fa0 //	slope Green =	10.747070
40132a80 40840838 //	slope Blue =	8.064209
40132a84 3fbffa03 //	slope Alpha =	127.953217
40132a88 201f0002 //	slope 1/w =	0.242188
40132a8c 42900000 //	area =	0.250000

Figure 2-25 The Triangle Engine Disassembly window.

Summary

Chapter 1, "Introducing the 3DO M2 Debugger," described the basic features of the 3DO M2 debugger and showed you how to start a debugging session. This chapter described the windows, menu items, and dialog boxes provided by the debugger, and showed how to use the debugger to run and debug your M2 applications.

Menu Reference

This appendix describes the menu items provided by the 3DO M2 debugger. The menus listed and described in this appendix are:

Topic	Page
The File Menu	52
The Edit Menu	54
The View Menu	55
The Execution Menu	58
The Target Menu	59
The Tools Menu	60

The File Menu

The File menu controls how the 3DO M2 debugger operates as an application in the Macintosh environment.

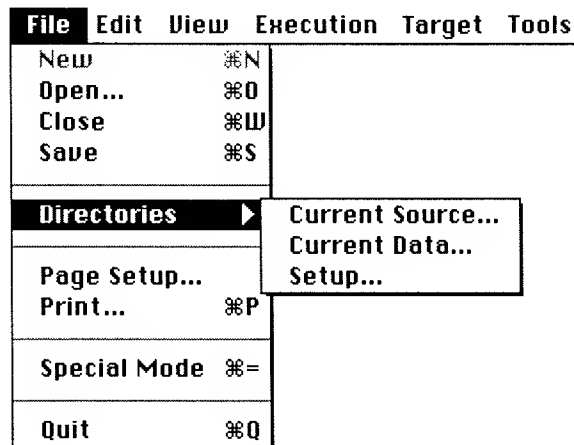


Figure A-1 File menu.

New

Does nothing and is always disabled.

Open (Command + O)

Displays a Standard File dialog box that allows the to view source files or other kinds of text files.

Close (Command +W)

Closes the active window. This menu item has same effect as selecting the Close box in the front window. It doesn't affect floating status window.

Save (Command + S)

Presents a Standard File dialog box that allows the user to save the contents of the topmost window. This menu item is implemented for the following windows:

- ◆ *Data Window*: Saves contents that are visible in window.
- ◆ *Disassembly Window*: Saves contents that are visible in window.
- ◆ *Portfolio Terminal*: Always enabled unless a Save operation has been performed and no changes have occurred since then in the window. This item saves contents of entire window.

-
- ◆ *PPC FP Register Window*: Saves contents of entire window.
 - ◆ *PPC User Register Window*: Saves contents of entire window.
 - ◆ *Symbols Window*: Saves contents of entire window.
 - ◆ *Triangle Disassembly Window* - Saves contents of entire window.

Directories

Displays the Directories submenu, which offers the commands listed in Table 2-3.

Table 2-3 *Subcommands of the Directories command.*

Command	Description
Current Source	Displays a dialog box containing one or more directory paths that the debugger searches to locate the appropriate source file.
Current Data	Displays a dialog box containing the directory path that the debugger will search to locate the appropriate symbol file.
Setup	Displays a CustomGetFile dialog box with a pair of checkboxes that allow the user to specify the directory path to the source files and/or data file.

Note: *If you have multiple source directories, you should create an .spt script file.*

Page Setup

Currently not implemented.

Print (Command + P)

Allows the user to print the contents of:

- ◆ *The Portfolio Terminal window*: Prints the contents of the entire window.
- ◆ *The Symbols window*: Prints the contents of the entire window.

Special Mode (Command + =)

Special mode is used to speed up target file I/O. When you turn on this mode, the Debugger no longer goes through `WaitNextEvent()` (which locks out all other Macintosh programs) but gives high priority to target file I/O. As a result, *printf* statements in your program run much faster than in normal mode. Target file I/O to */remote* is faster as well.

Special mode is useful at system startup, for example, to enable the target to get to the shell prompt more quickly.

To clear Special mode, press the space bar or click the mouse.

Quit (Command + Q)

Quits the debugger and returns to the Finder.

The Edit Menu

The Edit menu lets you edit text and display the Preferences dialog boxes.

Edit	View	Execution
Undo		⌘Z
Cut		⌘H
Copy		⌘C
Paste		⌘V
Clear		
Select All		⌘A
Preferences...		

Figure A-2 *Edit menu.*

Undo

Not implemented.

Cut (Command + X)

Removes selected text and puts it on the Clipboard, replacing the previous contents of the Clipboard. Implemented only in the Portfolio Terminal window, and only when text is selected.

Copy (Command + V)

Copies selected text to the Clipboard, replacing the previous contents of the Clipboard.

Paste

Inserts a copy of the Clipboard contents at the insertion point or replaces selected text.

Clear

Deletes selected text in the Portfolio Terminal window without copying it to the clipboard. Implemented only in the Portfolio Terminal, and only when text is selected.

Select All (Command + A)

Selects all text in the Portfolio Terminal window if it is the active window. Select All is always enabled, even if there's nothing to select. Implemented only in the Portfolio Terminal.

Preferences

No longer supported. Instructs the user to set preferences by choosing the Target | Setup menu item.

The View Menu

The View menu opens 3DO Debugger windows, brings existing windows to the front, and in some cases lets you open multiple windows of the same type.

All the windows listed in this section are described in more detail in Chapter 8, "Introducing the 3DO M2 Debugger," and Chapter 9, "Using the Debugger."

Besides being used to open windows, the View menu always displays a list of all currently open Variables, Source, Data, and Disassembly windows. This list appears at the bottom of the menu View menu. You can bring any one of these to the front by selecting it from the list. To open additional Variables, Source, Data, and Disassembly windows, select them from the body of the View menu. If any windows of these four types are left open when you exit from 3DODebugger, they are stored in the *3DODebugger.prefs* file and are reopened the next time 3DODebugger is launched.

Most View commands open the window that is selected if it is not already open, and bring it to the front if it is already open. There are four exceptions to this rule, however. When you select the Disassembly, Data, Source, or Variables menu item, an additional instance of the selected window opens if one or more windows of that type are already open.

View	Execution	Target	Tools
Disassembly		%J	
Data		%B	
Source		%K	
Variables		%M	
Symbols		%Y	
BreakPoints		%U	
Terminal		%T	
Task			
Status			
PPC User Registers			
PPC FP Registers			
PPC Supervisor Registers			
Stack Crawl			
Triangle Disassembly			
Send Variable Data		%D	

Figure A-3 *View menu.*

Disassembly (Command + J)

Opens a new Disassembly window.

Data (Command + B)

Opens a new Data window, displaying the contents of RAM starting at 0.

Source (Command + K)

Opens a new Source window (a text window for examining source code, setting breakpoints, and tracking control flow).

Variables (Command + M)

Opens a new Variables window. To bring an already open one to the front, select it from the list of open display windows at the bottom of the View menu.

Symbols (Command + Y)

Opens a new Symbols window, or brings it to the front.

BreakPoints (Command + U)

Opens a new BreakPoints window, or brings it to the front. A BreakPoints window is a text window for examining and altering current breakpoints

Terminal (Command + T)

Opens a new Portfolio Terminal window, or brings it to the front. The Portfolio Terminal window, often referred to simply as the Terminal window, is a text window for console I/O between the hardware and the user.

Task

Opens a new Task window.

Status

Opens a the Status window if it is not already open. The status window is a floating window for displaying information about the current task.

PPC User Registers

Opens a User Registers window, or brings it to the front. The User Registers window is a text window for examining and altering the contents of the general purpose registers.

PPC FP Registers

Opens an FP (floating point) Registers window, or brings it to the front. The FP Registers window is a text window for examining and altering the contents of the floating point registers.

PPC Supervisor Registers

Opens a Supervisor Registers window, or brings it to the front. A Supervisor Registers window is a text window for examining (but not altering) the contents of the Supervisor registers.

Stack Crawl

Opens a Stack Crawl window, or brings it to the front. The Stack Crawl window is a text window for examining the stack and displaying the path that was taken to arrive at the current level of scope.

Triangle Disassembly

Opens a Triangle Engine Disassembly window, or brings it to the front. The Triangle Engine Disassembly window is a text window for examining instructions going to the Triangle Engine.

Send Variable Data (Command + D)

Lets you pass variables from the Source window (as well as most text fields) to the Variables window. To do this, select the variable name and then choose View | Send Variable Data.

The Execution Menu

The Execution menu controls the execution of the program running on the 3DO M2 development (dev) card.

Execution	Target	Tools
Go		⌘G
Stop		⌘.
Kill Stopped Task		
Step Over		⌘,
Step In		⌘;
Ignore DebugBreakpoint		
Ignore breakpoints		
✓ Initial breakpoint in task		

Figure A-4 Execution menu.

Go (Command + G)

Resumes execution of a stopped program.

Stop (Command + .)

Stops target execution.

Kill Stopped Task

Removes a stopped task from the Task list and recovers any storage allocated to that task.

Step Over (Command + ,)

Causes a stopped task to resume execution, stopping at the next line of source in the current scope of the Source Window, or at the next instruction in the current scope of the Disassembly Window. If the task is stopped at a function call, the task does not stop until it returns from the function.

Step In (Command + ;)

Causes a stopped task to resume execution, stopping at the next line of source, even if it involves branching into a function and stopping there. If the Disassembly Window is frontmost it stops at the next instruction, rather than stopping at the next line.

Ignore DebugBreakPoint

When checked, this menu item causes the current task to continue executing even when it encounters a hard-coded breakpoint (DebugBreakpoint).

Ignore BreakPoints

Ignores software breakpoints (including any breakpoints that may be placed or cleared from within 3DODebug itself).

Initial BreakPoint in task.

When checked, this menu item causes the current task to stop at the first line in the task; that is, at the beginning of the task's `main()` function.

The Target Menu

The Target menu lets you set target preferences, restart the Debugger, and dump target RAM as a binary image file.

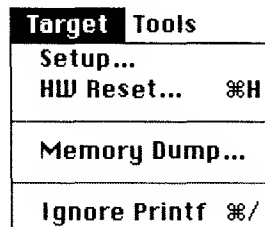


Figure A-5 Target menu.

Setup

Displays a dialog box that allows the user to specify the Target, ROM and Video, as well as the OS version to use (3DODebug selects the remote folder that occupies the same directory as the boot script).

HW Reset (Command + H)

Causes the target (M2 Dev Card) to reboot.

Memory Dump

Displays a sequence of dialog boxes that allows the user to select a range of memory which may be saved to a file as raw data. The resulting file may be opened and read in Resorcerer.

Ignore Printf (Command + /)

When checked, this menu item disables standard output of *printf* statements.

The Tools Menu

The Tools menu provides items that let you enable or disable file- system tracing and allow you to create a log of data access to help you optimize the placing of tracks on a CD-ROM



Figure B-26 *Tools menu.*

FileSystem Trace

When enabled, FileSystem Trace displays information at the Portfolio Terminal window about file-system transfers. Although FileSystem Trace is not necessary for typical debugging, it is a useful diagnostic tool.

CD Access Log

During the development stage of a title, this menu item creates a log of data accesses that can be used later to determine optimum strategies for laying out game tracks on a CD-ROM. When a CD access log has been created, it can be used to optimize CD-ROM accesses before the game is committed to CD-ROM. For more details, see the *3DO M2 CD Mastering Guide*.

Index

Symbols

symbol DBG-5, DBG-19
.spt file DBG-22
/remote> prompt DBG-8, DBG-9
-> symbol DBG-19, DBG-28

Numerics

3DO
 3dodebug folder DBG-8
3DO Debugger DBG-vii
 examining structures DBG-32
 Execution menu DBG-58
 File menu DBG-52
 preparing for launch DBG-2
 Quick Start DBG-vii, DBG-1, DBG-25
 Source window DBG-18
 Special Mode DBG-53
 starting DBG-2
 Target menu DBG-59
 Variables window DBG-33
 View menu DBG-55
3DO debugger icon DBG-8
3DO Development Environment Installation Guide
DBG-viii
3DO M2 debugger DBG-1
3DO M2 debugger, see also debugger DBG-1
3DO M2 debugger, see also M2 debugger DBG-1
3DODebug DBG-1
3DODebug, see also debugger DBG-2
3DODebug, see also M2 debugger DBG-2
3DODebug.Prefs file DBG-13
3DODebug.Prefs DBG-6

3DODebug.prefs DBG-55

A

active window
 closing DBG-52
arrays DBG-34
 working with DBG-38
asterisk (*)
 in Disassembly window DBG-20

B

Breakpoint
 removing DBG-29
 setting DBG-29
breakpoint
 clearing DBG-19, DBG-20
 initial DBG-16
 setting DBG-28
breakpoint symbol DBG-19
breakpoints DBG-vii
 in disassembly window DBG-20
 using DBG-27
Breakpoints command DBG-28, DBG-29
BreakPoints menu item DBG-56
BreakPoints window
 removing a breakpoint DBG-29
 setting a breakpoint DBG-29
 using DBG-29
Breakpoints window DBG-28, DBG-56
 experimenting with DBG-29

C

- CD Access Log menu item DBG-60
- cd command DBG-14
- Clipboard DBG-54
- Close menu item DBG-52
- Command + comma DBG-29
- Command + D DBG-36
- Command + G DBG-28
- Command + J DBG-19
- Command + M DBG-32
- Command + Return DBG-8
- Command + semicolon DBG-29
- Command + T DBG-9
- Command + U DBG-29
- commands
 - MPW DBG-9
- comment symbol (#) DBG-5, DBG-6
- conventions
 - typographical DBG-viii
- CR register DBG-44
- Current Data submenu DBG-53

D

- data
 - examining DBG-40
- Data menu item DBG-56
- Data window DBG-40, DBG-56
 - using DBG-40
- debug command DBG-14, DBG-15
 - arguments to DBG-15
- debug version of an M2 program DBG-5
- DEBUG_OPTIONS DBG-5
- debugger DBG-vii
 - components of DBG-2
 - features of DBG-1
 - introducing DBG-1
 - languages DBG-vii
 - loading DBG-8
 - M2 DBG-vii
 - setting up DBG-2
 - starting DBG-8
 - using DBG-vii
- debugger icon DBG-8
- debugger preferences DBG-6
- debugger preferences file, see also preferences file DBG-2
- debugger script DBG-6, DBG-7

- debugging session DBG-1
 - resuming DBG-26
 - setting up DBG-2
 - starting DBG-13
- developer (dev) card DBG-2
- development (dev) card
 - booting DBG-6
- development card DBG-9
- dialog boxes
 - Directories | Setup DBG-11
 - target setup DBG-6
- directories
 - configuring DBG-4
- Directories menu item DBG-53
- Directories submenu DBG-53
- Directories | Setup dialog box DBG-11
- directory
 - changing DBG-9
- Disassembly menu item DBG-56
- Disassembly window DBG-19, DBG-56
 - clearing breakpoints DBG-21
 - features of DBG-20
 - opening DBG-21
 - popup menu in DBG-21
 - setting breakpoints DBG-21
 - setting breakpoints in DBG-20
 - setting PC DBG-21

E

- editing text DBG-54
- Enter key DBG-9
- environment variables
 - setting DBG-12
- Evaluate button
 - in Variables window DBG-34
 - using DBG-34
- Evaluate text box
 - in Variables window DBG-39
 - using DBG-34
- examining data DBG-40
- examining structures DBG-32
- executable DBG-10
- Execute | Step In menu command DBG-29
- Execute | Step Over menu command DBG-29
- Execution menu DBG-58
 - menus
 - Execution DBG-16

Execution | Initial breakpoint in task menu item
menu items

Execution | Initial breakpoint in task
DBG-16

F

f symbolic information DBG-5
file

preferences DBG-2

File menu DBG-10, DBG-52

File | Find utility DBG-15

File | New menu item DBG-52

files

locations of DBG-10

storing in 3DODebug DBG-55

FileSystem Trace menu item DBG-60

Flash ROM DBG-6

floating-point registers window DBG-57
folder

newview DBG-4

FP (floating point) Registers window DBG-57

FP Registers window DBG-45

FP Supervisor Registers window DBG-47

G

-g build option DBG-5

-g option DBG-5

Getting Started With 3DO M2 Release 2.0 DBG-3

Go DBG-58

Go menu item DBG-58

H

hashmark (#) DBG-19

help window DBG-9

HW Reset menu item DBG-59

I

icon

3DO debugger DBG-8

Ignore Breakpoints menu item DBG-59

Ignore DebugBreakpoint menu item DBG-59

Ignore Printf menu item DBG-60

initial breakpoint DBG-16

Initial Breakpoint in task menu item DBG-59

K

Kill Stopped Task menu item DBG-58

killtask command DBG-17

L

languages DBG-vii

loading the debugger DBG-8

M

M2 debugger

features of DBG-vii

M2 debugger, see also debugger DBG-vii

M2 development card, see also development (dev)
card DBG-3

M2 development system DBG-vii

Macintosh DBG-viii, DBG-1, DBG-2, DBG-3

Macintosh operating system DBG-viii

Macintosh Quadra DBG-viii

main() function DBG-16, DBG-17, DBG-19,
DBG-28

makefile DBG-4

newview DBG-12

Memory Dump menu item DBG-60

Memory Dump window DBG-40

menu commands DBG-51

Execute | Step In DBG-29

Execute | Step Over DBG-29

View | Breakpoints DBG-29

View | FP Registers DBG-45

View | PPC Supervisor Registers DBG-47

View | PPC Users DBG-44

View | Stack Crawl DBG-47

View | Task DBG-43, DBG-44, DBG-45,
DBG-47

View | Triangle Disassembly DBG-48

View | Variables DBG-39

menu items DBG-51-??

Target | Memory Dump DBG-42

Target | Setup DBG-7

View | Send Variable Data DBG-36

View | Status DBG-17

View | Terminal DBG-9

View | Variables DBG-32

menu reference DBG-51-??

menus DBG-51

File DBG-10

monitor DBG-2, DBG-3
MPW
 and Portfolio Terminal window DBG-9
MPW commands DBG-9
MPW shell commands DBG-9
MPW Worksheet window DBG-9

N

New menu item DBG-52
newview application DBG-26
newview executable DBG-10, DBG-15
newview file DBG-4
newview folder DBG-4
newview makefile DBG-12
newview program DBG-3, DBG-14
 setting up debugger for DBG-10
newview.c DBG-3, DBG-4, DBG-12, DBG-14,
DBG-26
newview.make DBG-4
NTSC DBG-7

O

Objects folder DBG-5
operating system
 Macintosh DBG-viii

P

PAL DBG-7
PC indicator (->), see also program counter DBG-31
Portfolio Terminal DBG-57
Portfolio Terminal window DBG-55
Portfolio Terminal window DBG-27
 and MPW DBG-9
 described DBG-9
 using as a help window DBG-9
Power Macintosh DBG-viii
Power PC FP Registers window DBG-45
Power PC registers, see registers
Power PC Supervisor Registers window DBG-46
Power PC User Registers window DBG-44
 using DBG-44
PPC FP (floating-point) Registers window DBG-43
PPC FP Registers menu item DBG-57
PPC Supervisor Registers menu item DBG-57
PPC Supervisor Registers window DBG-43
PPC User Registers menu item DBG-57

PPC User Registers window DBG-43, DBG-44
PPC Users Registers window DBG-44
preferences
 debugger DBG-6
 setting DBG-6
Preferences dialog box DBG-54
preferences file DBG-2
preparing for DBG-1
printf statements DBG-60
program counter symbol (->) DBG-19
program-counter indicator (->) DBG-28
prompt
 remote> DBG-9

Q

Quadra DBG-viii

R

Receive Data DBG-34
registers DBG-vii
 modifying contents of DBG-44
 viewing contents of DBG-44
Registers User window
Registers windows DBG-43
remote> prompt DBG-9
removing a breakpoint
 BreakPoints window DBG-29
requirements
 system DBG-vii, DBG-viii
RotateScene() function DBG-28

S

Save menu item DBG-52
script
 debugger DBG-6
script, see also debugger script DBG-7
SDF (scene description format) file DBG-14
see also M2 debugger DBG-vii
Send Variable Data menu item DBG-58
setting a breakpoint
 Breakpoints window DBG-29
setting PC
 Disassembly window DBG-21
Setup menu item DBG-59
Setup submenu DBG-53

shell commands
 MPW DBG-9
 source files DBG-10
 Source menu item DBG-56
 Source window DBG-18, DBG-19, DBG-28, DBG-56, DBG-58
 introduced DBG-18
 opening DBG-19
 source-line symbol (#) DBG-19
 Special Mode DBG-53
 Special mode DBG-53
 Special Mode menu item DBG-53
 spt file, see .spt file DBG-22
 Stack Crawl menu item DBG-57
 Stack Crawl window DBG-47, DBG-57
 Stack window DBG-47, DBG-48
 Standard File dialog box DBG-52
 Status menu item DBG-57
 Status window DBG-17, DBG-18, DBG-57
 step commands DBG-29
 Step In DBG-59
 Step In command DBG-29
 using DBG-30
 Step In menu item DBG-59
 Step Over DBG-58
 Step Over command DBG-29
 using DBG-31
 Step Over menu item DBG-58
 stepping through code DBG-29
 Stop menu item DBG-58
 Supervisor Registers window DBG-46, DBG-57
 symbol file DBG-10
 symbolic-information file DBG-10
 Symbols menu item DBG-56
 Symbols window DBG-56
 System 7.1 DBG-viii
 system requirements DBG-vii, DBG-viii

T

Target menu DBG-59
 target monitor program DBG-2
 Target Setup dialog box DBG-6
 Target! Memory Dump menu item DBG-42
 Target! Setup menu item DBG-7
 Task menu item DBG-57
 Task window DBG-57
 using DBG-43
 Terminal menu item DBG-57

Terminal Portfolio window
 starting a debugging session DBG-16
 Terminal window
 described DBG-9
 Terminal window, see Portfolio Terminal window
 DBG-9
 texcow.csf DBG-26
 texcow.csf file DBG-15
 text
 deleting DBG-55
 editing DBG-54
 selecting DBG-55
 this DBG-24
 Tools menu DBG-60
 Triangle Disassembly menu item DBG-57
 Triangle Engine Disassembly window DBG-48, DBG-57
 typographical conventions DBG-viii

U

user interface DBG-1
 user registers DBG-44
 User Registers window DBG-43, DBG-44, DBG-57
 UTF (unified texel format) file DBG-14

V

Value field DBG-37
 Value text box DBG-37
 variable
 changing the value of DBG-34, DBG-36
 determining the value of DBG-34
 evaluating DBG-34
 examining the value of DBG-34, DBG-35
 variables
 working with DBG-32
 Variables menu item DBG-56
 Variables window DBG-32, DBG-36, DBG-37, DBG-56, DBG-58
 adding to display DBG-33
 and arrays DBG-38
 changing values interactively DBG-33
 deleting a variable DBG-33
 dereferencing a pointer DBG-34
 features of DBG-33
 selecting for receiving data DBG-34
 two ways to use DBG-34
 working with DBG-33

Variables windows

 multiple DBG-33

View menu DBG-55

View | Breakpoints menu command DBG-29

View | FP Registers menu command DBG-45

View | PPC Users menu command DBG-44

View | Send Variable Data menu item DBG-36

View | Stack Crawl menu command DBG-47

View | Status menu item DBG-17

View | Supervisor Registers menu command
DBG-47

View | Task menu command DBG-43, DBG-44,
DBG-45, DBG-47

View | Terminal menu item DBG-9

View | Triangle Disassembly menu command
DBG-48

View | Variables menu command DBG-39

View | Variables menu item DBG-32

W

Worksheet window

 MPW DBG-9

X

XER register DBG-44



3DO M2 DataStreamer Programmer's Guide

Version 2.0 – May 1996

Copyright © 1996 The 3DO Company and its licensors.

All rights reserved. This material constitutes confidential and proprietary information of The 3DO Company. This documentation is subject to a license agreement with The 3DO Company and may be used only by parties to such agreement. Use by any other persons, and/or for any purpose not expressly authorized by the agreement, is strictly prohibited.

3DO's LICENSOR(S) MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. 3DO'S LICENSOR(S) DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME JURISDICTIONS. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

Other brand or product names are the trademarks or registered trademarks of their respective holders.

Contents

About this Document.....	DSG-xi
Audience.....	DSG-xi
How this Document is Organized	DSG-xi
Typographical conventions.....	DSG-xii

1

3DO DataStreamer Overview

For More Information	DSG-1
Why Use the DataStreamer?	DSG-2
Examples of Using the DataStreamer	DSG-2
Preparing a Streaming Application	DSG-3

2

Preparing and Playing a Simple Stream

For More Information	DSG-5
What are Stream Files?	DSG-6
Stream File Preparation Basics	DSG-6
Stream File Terminology	DSG-7
Analyzing Data and Making Trade-Offs.....	DSG-7
System Resource Limitations	DSG-8
Trade-Offs	DSG-8
Generating and Converting Data.....	DSG-10
Generating Data	DSG-10
Converting Data to Run-time Formats	DSG-11
Compressing Data	DSG-12
Creating Chunk Files	DSG-13
Creating a Stream File With the Weaver Tool.....	DSG-13

Creating a Weaver Script	DSG-14
Using the Weaver Tool	DSG-16
Examining the Output Stream	DSG-16
Playing Back a Stream	DSG-16
Data Types and Subscribers	DSG-17
Playback Example Applications	DSG-17

3

Inside Streaming: Basics

DataStreamer Threads and Data Flow	DSG-20
DataStreamer Threads	DSG-20
Buffer Overview	DSG-21
Playing a Stream—Overview	DSG-22
Allocating Data Buffers	DSG-22
Example: Reading the Stream Header and Allocating Data Buffers	DSG-23
Deciding on Stream Block Size	DSG-24
Creating a Message Port and Message Items	DSG-24
Example: Setting up a Message Port	DSG-24
The Message-Sending Mechanism	DSG-25
Launching the Required Threads	DSG-26
Initializing and Connecting Data Acquisition and Streamer Threads	DSG-27
Initializing and Connecting Subscriber Threads	DSG-28
Extra Initialization for the Audio Subscriber	DSG-29
Registering for End-of-Stream Notification	DSG-30
Starting the Streaming Process	DSG-31
Using Persistent Data With the DATA Subscriber	DSG-31

4

Inside Streaming: Control

Time and the DataStreamer	DSG-33
The Stream Clock	DSG-33
Changing Time within a Stream	DSG-34
Starting and Stopping	DSG-34
Jumping to a New Location	DSG-36

5

Stream Data Formats

For More Information	DSG-39
Some Background on MPEG	DSG-39

MPEG Audio	DSG-40
MPEG Video	DSG-40
MPEG Multiplex	DSG-41
Some Background on the Data Streamer	DSG-41
Markers	DSG-42
Subscriber Chunk Common Header Format	DSG-43
Stream Control Chunks (STRM)	DSG-43
MPEG Video Stream Header Chunk	DSG-44
MPEG Video Data Chunks (3 Chunk Subtypes)	DSG-44
Recommendations	DSG-45
MPEG Audio Stream Header Chunk	DSG-46
MPEG Audio Frame Chunks (3 Current and 9 Potential Subtypes)	DSG-47
HALT Chunk	DSG-48
GOTO chunk	DSG-48
STOP Chunk	DSG-50

6

Debugging and Optimization

Debugging with the DataStreamer Libraries	DSG-51
Minimizing Filler	DSG-52
Guidelines for Minimizing Filler	DSG-52
Using Appropriate Streamblock Size and Number of Buffers	DSG-53
Streamblock Size Restrictions	DSG-53
Dealing with Data Rate Spikes	DSG-53
Using More Buffers to Prevent Data Starvation	DSG-54
Selecting Thread Priorities	DSG-54
Recommended Order of Process Priorities	DSG-54
How Bad Priorities Can Stop Your Stream	DSG-56
Using Tracing to Understand Data Flow	DSG-56
How to Use Tracing	DSG-57

7

Creating New Subscribers

Introduction	DSG-61
Conceptual Background	DSG-62
Buffering	DSG-62
Logical Channels	DSG-62
Control Calls	DSG-62
Subscriber Messages	DSG-63

Implementing Subscriber Functionality	DSG-64
Subscriber Message Opcodes	DSG-65
Queued Data and Branching Operations	DSG-65
Video Data	DSG-66
Audio Data	DSG-66

List of Figures

Figure 2-1 Three chunk files are multiplexed to create a single output stream file.	DSG-6
Figure 2-2 Data-conversion flowchart.	DSG-11
Figure 2-3 Example for a basic Weaver script.	DSG-14
Figure 3-1 DataStreamer threads.....	DSG-20
Figure 3-2 Data streaming buffer flow process.....	DSG-22
Figure 4-1 Weaver script including stream control chunks.	DSG-36
Figure 5-1 Typical frame sequences	DSG-40
Figure 5-2 Chunk nesting.	DSG-48
Figure 6-1 Data demand spikes.	DSG-54
Figure 6-2 Raw trace example.	DSG-58
Figure 6-3 Trace log after Canon tool has been used.....	DSG-58
Figure 6-4 Using “Search” to filter a trace log	DSG-59

List of Tables

Table 2-1	DataStreamer terminology.....	DSG-7
Table 2-2	Guidelines for data generation.....	DSG-10
Table 2-3	Conversion tools.....	DSG-12
Table 2-4	Compression tools.....	DSG-12
Table 2-5	Data conversion tools.....	DSG-13
Table 2-6	Available data types, chunk types and subscribers.....	DSG-17
Table 2-7	Playback example applications.....	DSG-17
Table 3-1	Message header fields.....	DSG-26
Table 7-1	Subscriber message opcodes.....	DSG-65

Preface

About this Document

This document helps you understand the 3DO M2 DataStream. The first two chapters provide a general overview of the process of getting your synchronized data to the 3DO M2 system. Later chapters discuss the architecture of the 3DO M2 DataStream and provide information about debugging, optimizing, and customizing DataStream applications.

Audience

This document is for C programmers who are preparing titles for the 3DO M2 development system.

How this Document is Organized

This document is organized as follows:

- ◆ Chapter 1, "3DO DataStream Overview," provides a basic overview of the DataStream library and associated tools.
- ◆ Chapter 2, "Preparing and Playing a Simple Stream," describes moving your data from the Macintosh to synchronized playback on the 3DO system.
- ◆ Chapter 3, "Inside Streaming: Basics," discusses the different tasks participating in a DataStream application and how to initialize and use them.
- ◆ Chapter 4, "Inside Streaming: Control," discusses using control chunks to start and stop the stream and to allow user-controlled branching.
- ◆ Chapter 5, "Stream Data Formats," describes the data format used by the DataStream.
- ◆ Chapter 6, "Debugging and Optimization," gives a variety of useful information including tracing, using priorities, and minimizing filler.

- ◆ Chapter 7, "Creating New Subscribers," discusses creating your own subscriber.

Typographical conventions

The following typographical conventions are used in this document:

Item	Example
code example	<code>int32 OpenGraphicsFolio(void)</code>
procedure name	<code>CreateScreenGroup()</code>
new term or emphasis	That added weight is called a <i>fat</i> .
file or folder name	The <i>remote</i> folder, the <i>demo.scr</i> file.

3DO DataStreamer Overview

The 3DO DataStreamer is a set of data preparation tools and a run-time library of C code that allow retrieval and playback of time-based data. Typical uses include:

- ◆ Playing back a compressed movie and its sound track while reading it from a CD.
- ◆ Loading application-specific data while an audio-visual presentation is running.

This chapter looks at the following topics:

Topic	Page
Why Use the DataStreamer?	2
Preparing a Streaming Application	3

For More Information

The manpages for 3DO DataStreamer library calls are in Chapter 1, “DataStreamer Functions,” of the *3DO M2 DataStreamer Programmer’s Reference*.

Data streaming tools are discussed in detail in Chapter 2, “Data Streaming Tools,” of the *3DO M2 DataStreamer Programmer’s Reference*.

Why Use the DataStreamer?

This section illustrates when data streaming is useful by discussing the following topics:

- ◆ Examples of Using the DataStreamer
- ◆ Preparing a Streaming Application

Examples of Using the DataStreamer

The DataStreamer is useful for all applications that need to play back synchronized data of different types (for example, video, audio, animations) while reading it from a CD. If the data could fit in the RAM, a much smaller program could play it.

The DataStreamer consists of libraries, tools, and a number of examples. You can use these example programs with minimal modification to stream your data to the 3DO system. Here's a list of examples including names of relevant example programs:

- ◆ Play a full-screen digital video sequence with sound. Examples: VideoPlayer, EZFlixPlayer
- ◆ Play MPEG video synchronized with 3DO or MPEG audio. Example: VideoPlayer
- ◆ Play multi-channel audio. Example: PlaySA

More extensive additions to the available source code make it possible to stream any kind of data you need. For example, if you have a proprietary software compression algorithm, you can modify the DataStreamer to play back video compressed with your own codec, synchronized with sound.

Preparing a Streaming Application

The process of preparing a streaming application is described as follows:

1. Analyze your data and plan how to stream it. Memory, I/O, RAM and especially CD-ROM bandwidth limitations require trade-offs. For example, you may not be able to perform full-screen software video decompression and MPEG audio decompression at the same time. More importantly, you have to divide the CD-ROM bandwidth between video, audio, and background-loading (DATA) data. See "Analyzing Data and Making Trade-Offs," in Chapter 2.
2. Convert source data to 3DO formats and compress if appropriate. See "Converting Data to Run-Time Formats."
3. Create chunk files using the DataStreamer tools. See "Creating Chunk Files" in Chapter 2.
4. Merge (multiplex) all data into one stream file with the Weaver tool. See "Creating a Stream File with the Weaver Tool," in Chapter 2.
5. Run the stream file using an example playback application. See "Playing Back a Simple Stream," in Chapter 2.
6. Add procedure calls in your program to the Data Stream library and link it in.

After initial verification of your data, you may want to modify streaming according to your needs. This can include the following:

- ◆ Optimize the stream file and the application, making trade-offs where needed. Use tools for examining stream files and the tracing facilities for optimum use of bandwidth and other resources. See Chapter 6 "Debugging and Optimization."
- ◆ Add branching and user input capabilities (start, stop, and so on). See Chapter 4 "Inside Streaming: Control."

Preparing and Playing a Simple Stream

This chapter provides all of the information you need to play back your data using the DataStreamer library and one of the example programs. You first learn about data analysis and preparation and then about the different playback example applications and where to find them.

This chapter looks at the following topics:

Topic	Page
What are Stream Files?	6
Analyzing Data and Making Trade-Offs	7
Generating and Converting Data	10
Creating a Stream File With the Weaver Tool	13
Playing Back a Stream	16

For More Information

The *3DO M2 DataStreamer Programmer's Reference* provides manpages for all functions and tools, including the Weaver script commands.

Some of you will want to modify the DataStreamer library source code, or one of the example applications, to suit your particular needs. This is discussed later in this manual.

What are Stream Files?

The DataStreamer works on a stream file. The stream file contains all video, audio, and other data in a compressed and multiplexed form and also contains timestamps. This section explains these basic concepts:

- ◆ Stream File Preparation Basics
- ◆ Stream File Terminology

Stream File Preparation Basics

Preparing the stream file involves two steps:

1. Compressing data and creating *chunk files* of each type of data using the appropriate DataStreamer tools.
2. Using the Weaver tool with a Weaver script to create a single stream file from the different chunk files. The stream file contains:
 - ◆ All *data*, interleaved, with timestamps.
 - ◆ An optional *stream header* chunk, containing information about buffer size, audio channels, and so on, that the Weaver tool extracts from the Weaver script.
 - ◆ *Stream Control* and *Marker Table* chunks that mark branch locations and special actions (optional).

Figure 2-1 shows an example of three files that have been merged into a single stream file with a stream header. The numbers are the timestamps associated with each data chunk. In the output stream, the shaded areas are filler, placed in the stream to pad the data to the stream block size.

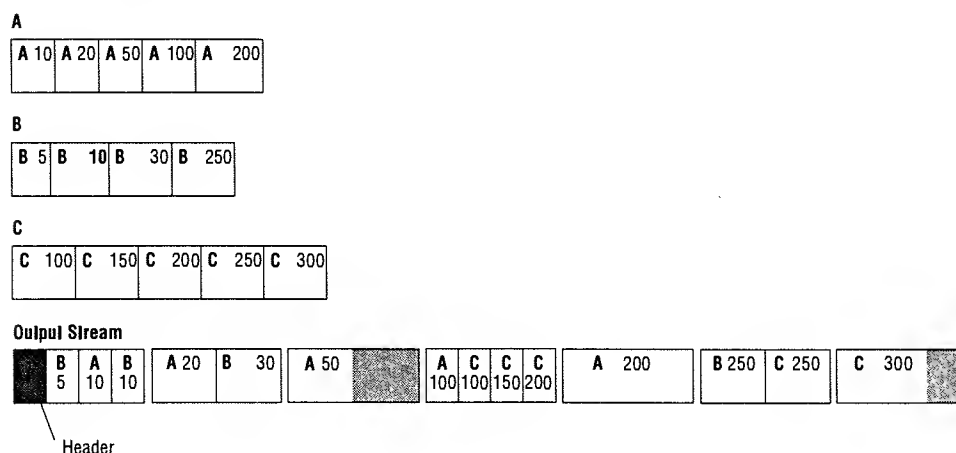


Figure 2-1 Three chunk files are multiplexed to create a single output stream file.

Stream File Terminology

To better understand the structure of a stream file, look at the definitions and explanations in Table 2-1.

Table 2-1 *DataStreamer terminology.*

Term	Definition	Detail
Stream file	A file that contains multiplexed data that the DataStreamer library can de-multiplex and display.	You create a stream file using the Weaver tool and a Weaver script.
Chunk	An atomic packet of data of one type; the basic element of a stream.	<p>Chunks are timestamped, some with duration. Types include:</p> <ul style="list-style-type: none"> ◆ EZFL (EZFLix) ◆ MPVD (MPEG video) ◆ STRM (stream control) ◆ SNDS (M2 audio) <p>For a complete list, see Table 2-6, "Available data types, chunk types and subscribers."</p>
Streamblock	Each stream is a sequence of streamblocks. Each streamblock is composed of data chunks (packets). Disc reads are done streamblock by streamblock.	A stream file's streamblock size must be a multiple of the device's block size.
Stream time (aka presentation time)	Every chunk has a stream time, the time at which the chunk's contents need to be presented.	Time is counted in audio folio ticks. Audio ticks are $44100/184 \approx 239.67\text{Hz}$.

Analyzing Data and Making Trade-Offs

A 3DO application has to work within the limitations of available memory and maximum data rate. Therefore, it is important to analyze data and to set clear goals early to have smooth playback later on. This is particularly important when using 2x CD drives. Even with 4x CD drives, it is still a useful exercise. As an example, for background music during an introduction, you'll want to use

CD-quality stereo, whereas sound files that consist only of people talking can be highly compressed and of lower quality. Data preparation can proceed once those decisions are made.

Note: *Be sure to analyze your data carefully and set your priorities. Some developers have had to process their data all over again because they made a wrong decision about optimal delivery early in their development cycle.*

This section looks first at the limitations of the hardware, and then at some trade-offs you have to make when designing your application.

System Resource Limitations

Consider the availability of system resources when preparing a stream file.

- ◆ **I/O bandwidth**—What percentage of available I/O bandwidth does your stream consume? In the case of the M2's quadra-speed CD-ROM drive, a stream can not exceed an average data rate of 600 kilobytes per second.
- ◆ **Other resources**—Availability of DSP (digital signal processing) resources for sound, and CPU cycles for video and MPEG audio decompression, can constrain playback.

If you approach the limits for I/O bandwidth (which may be rare with a 4x CD drive) and other system resources, creating a working stream becomes a balancing act.

For example, in a typical audio/video playback application, MPEG video bandwidth might be approximately 220 KB/sec. 2:1 compressed AIFC audio bandwidth might be 88KB/sec. Therefore, the total bandwidth required is at least 308 KB/sec. For a 4x CD drive, this leaves 292 KB/sec, minus a small amount of filler overhead.

Note: *The M2 Weaver can split an MPEG video chunk if it doesn't fit the current stream block, virtually eliminating filler in many situations.*

For more information

See Chapter 6, "Debugging and Optimization."

Trade-Offs

This section discusses issues you need to consider when preparing video and audio for your 3DO application.

Note: *An upper-case 'K' represents a value of 1024. A lower-case 'k' represents a value of 1000.*

Audio

- ◆ **Audio compression**—To stream true CD-quality audio (uncompressed 44.1 kHz, 16-bit stereo), you need 176 kB/sec. However, near CD-quality audio can be achieved at 88 kB/sec using 2:1 CBD2 AIFC compression. Near CD-quality audio can also be achieved at about 30kB/sec, using MPEG audio compression. See Table 2-2 for more information.

Video

- ◆ **Size of video**—You can achieve excellent quality video, at full frame size and rate, using MPEG-1 Video at 180-250 KB/s. This video signal can be combined with near CD-quality audio, using MPEG-1 Audio at 30-40 kB/sec, and still stay well under 300 KB/sec. Even without MPEG-1 audio, 3DO's 2:1-compressed 44.1 kHz stereo provides equivalent quality at 88 kB/sec.
- ◆ **Branching**— If you allow user interaction:
 - ◆ Make sure you have key frames at branching points. To find out where key frames (I frames) are located, look at the output of the `MPEGVideoChunkifier` utility.
 - ◆ If you need to branch quickly, use smaller buffers. Buffer size should be between 32 KB and 128 KB, and must be a multiple of the disc block size (2 KB). Larger buffers result in less stream bandwidth wasted on stream block filler. However, a stream with MPEG video will have very little filler.
- ◆ **Buffer allocation**—The streamer cannot refill a stream buffer until each chunk in that streamblock has been used. If you have data your application needs to use over a long period, this can cause problems. For example, you may have textures that change rarely and video that changes frequently. If you interleave the video with the textures, you can't reuse a streamblock containing a texture (including all of the space devoted to video chunks) until the texture is no longer in use. The way around this is using the `DATA` subscriber to copy data out of the stream buffer or to assign a large number of buffers.
- ◆ **Data rate issues**—To achieve smooth streaming, look at the data rate for each data type before weaving. If you add up the rates and they don't exceed 600 KB/sec, weave the stream and look at the data rate again (now that it contains filler). To look at the stream file, use `dumpstream`.

It may be acceptable to have a rate slightly above 600 KB/sec for brief periods of time. The `DataStream` can smooth over such a data rate spike, if it has enough buffer space. But you'll have to test the stream on a range of CD drives. Having a data rate that's consistently higher than 600 KB/s will inevitably cause problems.

Generating and Converting Data

This section describes how to generate and convert your data for optimal results and lists available tools. The process consists of four steps:

- ◆ Generating Data
- ◆ Converting Data to Run-time Formats
- ◆ Compressing Data
- ◆ Creating Chunk Files

Generating Data

Table 2-2 provides guidelines for generating video and audio data to achieve optimum performance.

Table 2-2 *Guidelines for data generation.*

Data type	Guidelines for generation
Video	Sample video at the highest possible quality and then compress it. Unless you are using a high-quality, real-time, MPEG encoder, be careful about using a tool that compresses video while digitizing—compressing video twice can be unsatisfactory. If video was generated from a movie with a Telecine process, use the 3-2 Pulldown tool to remove superfluous frames.
Artwork	Start artwork at a high resolution and then convert it. Remove NTSC hot colors.
Audio	Sample audio at 44.1 kHz and convert it to a 22.05 kHz sample rate if required. Deciding on the optimal compression may require some experimentation. Here's some information to help you decide on the compression rate: <ul style="list-style-type: none">◆ CD-quality stereo requires 176 kB/sec.◆ The lowest-quality sound currently offered (22.05 kHz Mono 4:1) is acceptable for voice in many cases, and requires 11 kB/sec.◆ Music without much high-frequency content can use 22.05 kHz, 16-bit stereo 2:1 compressed (44 kB/sec data rate).

For More Information

Data generation issues are discussed in more detail in *3DO M2 Tools for Sound Design*, *3DO M2 Graphics Tools* and the *3DO M2 Video Processing Guide*.

Converting Data to Run-time Formats

Before you can create chunk files from data, you have to convert them to a run-time format and compress them where appropriate. The M2 2.0 CD-ROM offers a variety of tools for converting still art, video, and audio. The related documentation provides detailed information on using those tools. Figure 2-2, and the note that follows, illustrate the various steps used to convert some common starting data formats into an M2-playable stream, containing MPEG-1 Video and 3DO Audio.

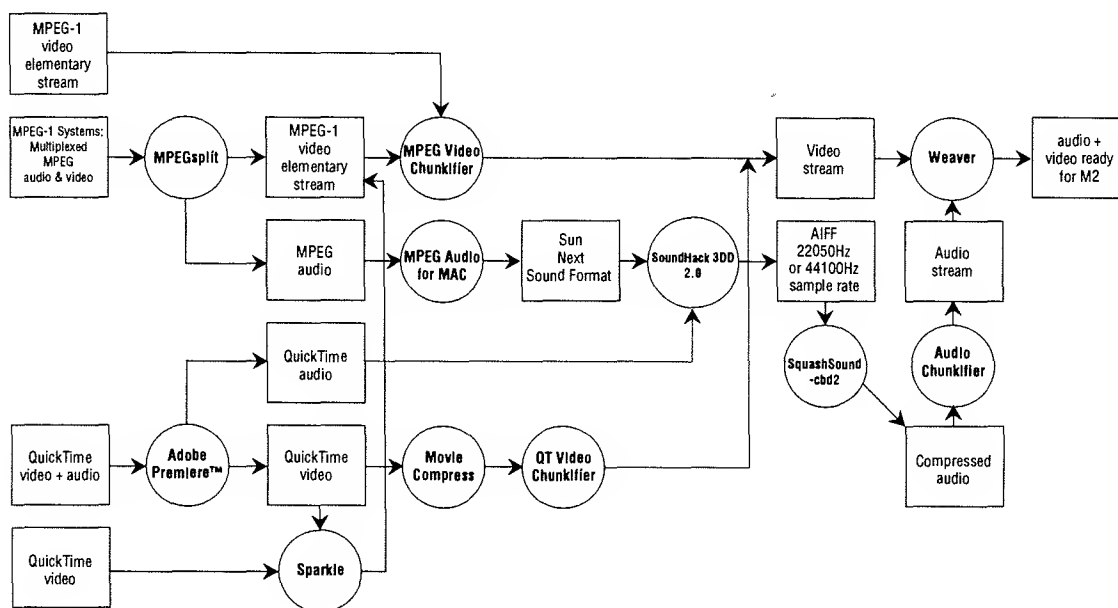


Figure 2-2 Data-conversion flowchart.

Note: The direct input formats to the MPEGVideoChunkifier and AudioChunkifier are MPEG-1 Video elementary stream, and AIFF audio. Any other formats must first be converted to these.

Note: Not all of the tools shown in Figure 2-2 are provided by 3DO.

Note: Figure 2-2 does not show all M2-supported stream formats (such as EZFlx or MPEG-1 Audio).

Table 2-3 lists the procedures and tools for converting different kinds of data.

Table 2-3 *Conversion tools.*

Data type	Guidelines for conversion
Still photo	Convert a PICT file to an MPEG still image using Sparkle.
Video and animation	<p>Convert a QuickTime video to a compressed movie using the following (or equivalent) tools:</p> <ul style="list-style-type: none">◆ 3:2 Pulldown if the movie was based on a film◆ MovieEdit to fine-tune the movie◆ MovieCompress for EXFlix compression, or MPEGXpress or Sparkle for MPEG compression
Sound	<p>Use Adobe Premier to extract an AIFF file from a movie (select either 44100 Hz or 22050 Hz sample rate).</p> <p>Use SquashSound or 3DO SoundHack (if necessary) to down-sample 44.1 KHz files to 22.05 kHz AIFF, compress if desired using the SquashSound utility.</p> <p>If you work with a Macintosh sound resource, you may need to resample in order to generate a 22.05 kHz AIFF file.</p>

Compressing Data

Compression can sometimes be done as part of the data conversion process. For example, when you convert sound to the 3DO format using the SoundHack tool, you can choose between 2:1 and 4:1 compression. Table 2-4 illustrates your choices.

Table 2-4 *Compression tools.*

Data type	Tool
Video	<p>You have two choices for compressing video:</p> <ul style="list-style-type: none">◆ EZFlix (3DO proprietary compression algorithm)◆ MPEG-1 Video
Audio	<p>The 3DO SquashSound tool offers 2:1 (SQS2 or CBD2) or 4:1 (AD-PCM). Note that SQS2 (one of the compression types produced by SquashSound) only works for mono.</p>

Creating Chunk Files

After data has been converted to a run-time format, it needs to be converted to *chunk* files. “Chunks” are the units of data delivered to the subscriber. The tools shown in Table 2-5 convert data to chunked, time-stamped files that the Weaver tool can then weave (multiplex) into a stream.

Table 2-5 *Data conversion tools.*

Tool	Description
QTVideo-Chunkifier	Converts QuickTime movie, that has been EZFlix-compressed with a QuickTime tool (such as MovieCompress), into a chunk stream for <i>EZFlix subscriber</i> . Note that <i>QTVideoChunkifier</i> only deals with the video data; you must extract and process any audio data from the QuickTime movie separately.
AudioChunkifier	Converts AIFF/AIFC data into a chunk stream for <i>SAudioSubscriber</i> .
MPEGVideo-Chunkifier	Converts an MPEG-1 video elementary stream into a chunk file for <i>MPEGVideoSubscriber</i> .
MPEGAudio-Chunkifier	Converts an MPEG-1 audio elementary stream into a chunk file for <i>MPEGAudioSubscriber</i> .
DATAChunkify	Converts a data file, that has been optionally compressed with the <i>Comp3DO tool</i> , into a chunk stream for <i>DATASubscriber</i> .

For More Information

Chapter 2, “Data Streaming Tools,” in the *3DO M2 DataStreamer Programmer’s Reference*, provides manpages for each data streaming tool.

Creating a Stream File With the Weaver Tool

Because the CD Drive can only read one file at a time, the DataStreamer processes only one stream file at a time. This means that you have to multiplex, or *weave*, the chunk files into a stream file that is suitable for playback. To create a stream file: prepare a *Weaver script file* and then submit that script and the chunk files to the *Weaver tool*.

This section describes the operations that you perform when creating the stream file:

- ◆ Creating a Weaver Script
- ◆ Using the Weaver Tool

◆ Examining the Output Stream

Creating a Weaver Script

The Weaver tool uses an expedient little scripting language to control the assembly of chunk files into streams. Some of the information in the script determines the order of the chunks. Other information generates a stream header at the front of the stream file.

Basic Weaver Script

A weaver script consists of command lines that are the “instructions” to the Weaver to build a stream file from the different chunk files. Each line in a Weaver script consists of a command and, zero, one, or more arguments.

Figure 2-3 shows a Weaver script that generates a file for use with the VideoPlayer example application inside the *Streaming* folder on the M2 2.0 CD-ROM. Note that not all the commands used in this sample script are necessarily needed by all applications.

The Weaver operates by first parsing the entire weaver script to gather all of the parameters. Then it outputs the stream header chunk, if requested. Finally, the Weaver reads chunks from the specified input files outputs them in time stamp order.

Some additional script commands are also discussed in “Using Weaver Script Command to Place Markers,” in Chapter 4.

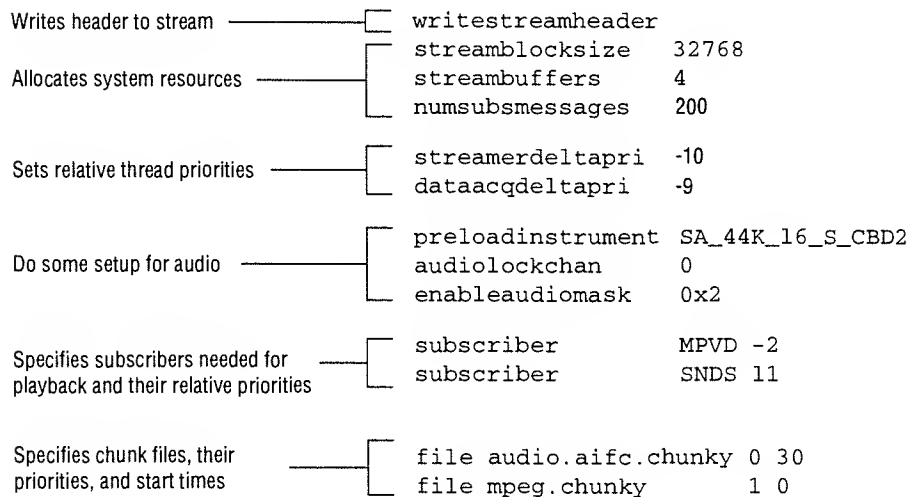


Figure 2-3 Example for a basic Weaver script.

Here's an explanation of the commands used in the example script:

- ◆ **writestreamheader**—writes a stream header chunk (with information obtained from the weave script) to the front of the stream file.
- ◆ **streamblocksize**—specifies the size of the stream blocks. Since stream blocks and playback buffers are the same size, it also specifies the buffer size.
- ◆ **streambuffers**—specifies how many CD I/O buffers the data streamer application should allocate for stream playback.
- ◆ **numsubsmessages**—specifies how many subscriber messages the data streamer should allocate. This number should be significantly larger than the total buffer size ((streambuffers * stream block size)/average chunk size). If the streamer runs out of subscriber messages it will abort playback.
- ◆ **streamerdeltapri** and **dataacqdeltapri**—specifies the priority for the streamer and data acquisition threads, relative to your application. The threads are discussed in detail in "DataStreamer Threads," in Chapter 3.
- ◆ **preloadinstrument**—specifies an instrument to preload for playing audio at a later time. Preload instruments so that the CD drive doesn't have to seek to load them during playback (possibly causing a playback hiccup).
- ◆ **audioclockchan**—specifies the channel that drives the streamer clock. This channel must have data whenever the stream contains time-dependent data.
- ◆ **enableaudiomask**—specifies the channels enabled for audio at the beginning of the stream. This is necessary if you use audio in several channels.
- ◆ **subscriber**—specifies which subscribers to launch in order to play back the stream, and their priorities relative to the application.
- ◆ **file**—specifies the name of a chunk file or stream file and how to multiplex it into the output stream. The priority argument controls how the Weaver tool places chunks with identical timestamps into the output stream. The start-time argument specifies the output stream time at which the file's first data should be delivered to its subscriber.

For More Information

The brief discussion of the Weaver script in this chapter is supplemented by more detailed information in later chapters. A manpage for each Weaver script command is included in the *3DO M2 DataStreamer Programmer's Reference*.

There are sample Weaver scripts on the M2 2.0 CD-ROM which include code to help you get oriented.

Using the Weaver Tool

The Weaver is an MPW tool that expects a script file and a number of data files as input and writes out a single stream file. Here's an example of invoking the Weaver tool with a script called *NewMovieScript* that weaves a video and an audio file:

```
weaver -o MyNewStream < NewMovieScript
```

Examining the Output Stream

During the stream-creation process, it is often helpful to look at the contents of a stream. This section discusses the basic stream format and how to examine a woven stream file.

Weaver Output—Basic Stream Format

A stream file created by the Weaver has the following basic format:

- ◆ File is a series of fixed-size blocks containing chunks
- ◆ File starts with a header chunk giving the stream parameters
- ◆ Each data chunk consists of a small descriptor followed by data
- ◆ The chunk descriptor contains fields identifying:
 - ◆ data type and subtype
 - ◆ presentation timestamp
 - ◆ chunk size
 - ◆ channel number

Available Tools

The `dumpstream` MPW tool analyzes the contents of a woven stream and prints out a listing of those contents. Here's an example:

```
DumpStream -v MyNewStream
```

See the *3DO M2 DataStreamer Programmer's Reference* for more information.

For More Information

Chapter 6, *Debugging and Optimization*, provides more information about optimization techniques for stream files, such as reducing filler.

Playing Back a Stream

After you've converted your data and created a stream file, you are ready to play it. The *Examples: M2_2.0: Streaming: Tools_and_Data* folder contains a variety of examples that you can use or modify to play a stream file with your own data.

Data Types and Subscribers

For each type of data (video, text, and so on) you want to stream to the M2 development system, your program needs to launch the appropriate subscriber. Table 2-6 provides an overview of different types of data, their chunk type identifiers, and the subscriber used to play each.

Table 2-6 Available data types, chunk types and subscribers.

Type of data	Chunk type	Subscriber
video	EZFL	EZFlix Subscriber
	MPVD	MPEG Video Subscriber
audio	SNDS	SAudio Subscriber
stream control	MPAU STRM	MPEG Audio Subscriber (handled directly by the data stream thread)
data	DATA	DATA Subscriber

Playback Example Applications

Table 2-7 lists example applications you can use to stream different kinds of data. You can examine them to better understand data streaming and modify them to suit your own data.

Table 2-7 Playback example applications.

Example	Description	Location
VideoPlayer	Plays MPEG video and either 3DO or MPEG audio	Tools and Data folder
PlaySA	Plays 3DO or MPEG audio. It's possible to have different audio in different channels.	Tools and Data folder
EZFlix Player	Plays EZFlix Video and 3DO audio	Tools and Data folder

Note: See the worksheet in the Tools_and_Data folder for data stream preparation scripts.

Inside Streaming: Basics

This chapter provides an overview of the architecture of the DataStreamer library, then discusses how a program can use the library by looking at the *VideoPlayer* example.

Since optimizing your streaming application requires some understanding of the streaming process, the optimization chapter follows the “Inside Streaming” chapters.

This chapter discusses the following topics:

Topic	Page
DataStreamer Threads and Data Flow	20
Playing a Stream—Overview	22
Creating a Message Port and Message Items	24
Launching the Required Threads	26
Extra Initialization for the Audio Subscriber	29
Registering for End-of-Stream Notification	30
Starting the Streaming Process	31
Using Persistent Data With the DATA Subscriber	31

DataStreamer Threads and Data Flow

The DataStreamer uses several threads as data-flow processing modules to continuously stream data from the CD Drive, through decompressors to the output logic, as synchronized playback. This section gives a conceptual overview of the architecture by looking at:

- ◆ DataStreamer Threads
- ◆ Buffer Overview

DataStreamer Threads

The following threads are discussed in this section:

- ◆ Data Acquisition Thread
- ◆ Streamer Thread
- ◆ Subscriber Threads

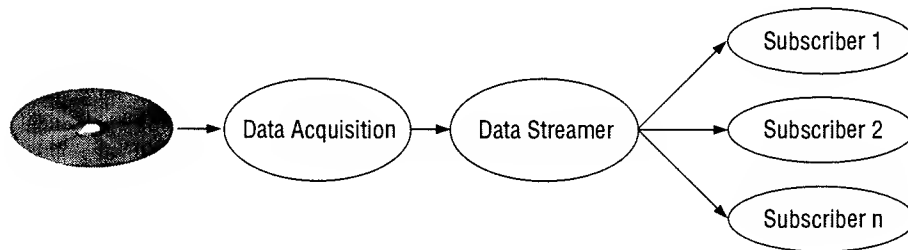


Figure 3-1 *DataStreamer threads.*

Data Acquisition Thread

A *data acquisition* thread supplies the streamer thread with data from a CD. An application can switch data acquisition threads at run-time to switch between stream files.

Streamer Thread

The *streamer thread* accepts fixed-sized blocks of data from the acquisition thread, demultiplexes them into chunks, and delivers the chunks to their respective subscribers. The streamer thread is responsible for data flow control at the block level, for the stream presentation clock, for controlling the data acquisition and subscriber threads, and for interacting with the client application. Note that data is delivered by reference, not by copying.

Subscriber Threads

A *subscriber thread* works on the data chunks that the streamer thread delivers. Each subscriber thread accepts and decompresses only one type of data, although it can accept multiple “channels” of the same type (for example, audio channels in English and French). The application interacts with the subscribers to get the decompressed data. The only exception are audio subscribers, which deliver their data directly to the sound spooler for playback. The number of subscribers depends on how many different types of data you want to play back.

The video subscribers hand frames over to the application. The playback application’s main thread of execution controls the screen and displays the frames.

Buffer Overview

Data streaming continuously performs the following steps (see Figure 3-2).

1. The data acquisition thread receives empty buffers from the streamer thread, pre-allocated by the playback application.
2. The data acquisition thread fills the buffers with data blocks from the CD. At this point, the different chunk types (audio, video, etc.) are interleaved and ordered based on timestamp and priority information. The data acquisition thread passes the data buffers back to the streamer thread.
3. The streamer thread demultiplexes each filled buffer and distributes each chunk, by chunk type, to the subscriber thread that is registered for that chunk type.
4. When a subscriber thread has played or is otherwise finished with a data chunk, it “returns” access rights to the chunk back to the streamer thread.
5. After all chunks in a buffer have been returned to the streamer thread, it sends the now-empty buffer back to the data acquisition thread to be filled with more data (see Step 2).

These steps continue until the data acquisition thread indicates an end-of-file condition or the stream is stopped by the client application.

Modifications to this basic flow are possible:

- ◆ At any time, the playback application can ask the streamer to seek (or “branch”) to a new position in the stream file.
- ◆ The playback application can connect a different data acquisition thread to the streamer to supply data from a different source. This disconnects the previous data acquisition thread.

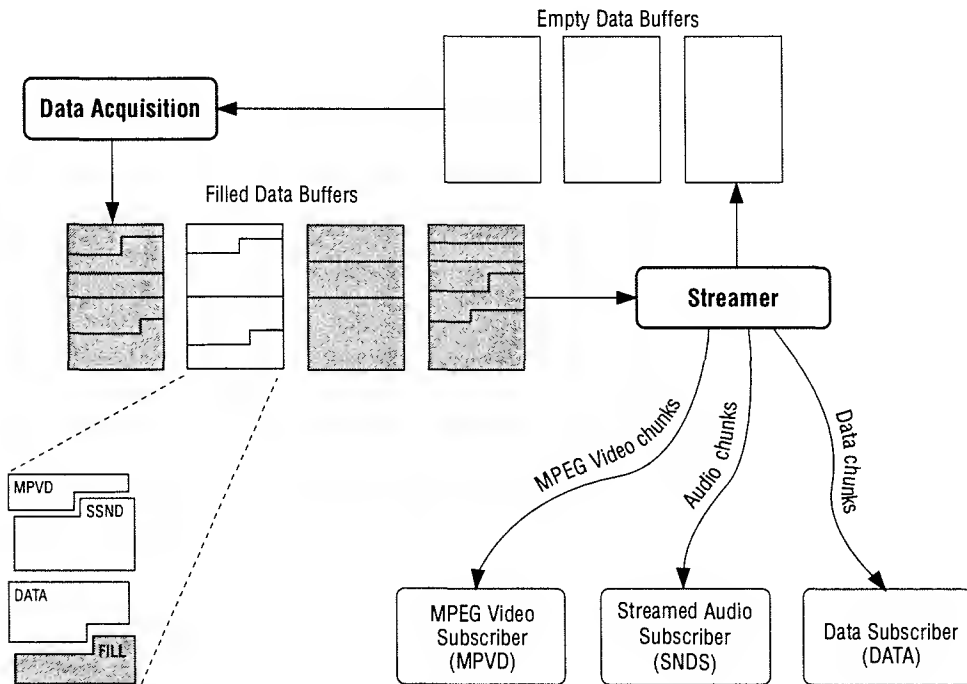


Figure 3-2 Data streaming buffer flow process.

Playing a Stream—Overview

To play a stream file on the 3DO M2 system, an application has to perform the following actions:

- ◆ Allocate data buffers
- ◆ Create a message port and a message
- ◆ Launch the required threads
- ◆ Register for end-of-stream notification
- ◆ Start the streaming process

The following sections look at each of these activities in some detail, using examples from the sample code provided on the M2 2.0 CD-ROM.

Allocating Data Buffers

This section looks at sample code that creates data buffers used by an application using the DataStreamer. It also briefly looks at some heuristics for deciding on buffer size.

Example: Reading the Stream Header and Allocating Data Buffers

A program that wants to use the streamer needs to allocate the stream buffers, using the number of stream buffers and the streamblock size from the stream file's stream header. The stream header was written based on information in the Weaver script. This is illustrated in Example 3-1, taken from *InitVideoPlayerFromStreamHeader()*, in *VideoPlayer/playvideostream.c*.

Example 3-1 *Reading the stream header and creating the buffer list.*

```

/* Get the stream header loaded */
status = FindAndLoadStreamHeader(&ctx->hdr, streamFileName);
/* If there was no stream header then use a default */
if ( status == kDSHeaderNotFound )
    status = UseDefaultStreamHeader(&ctx->hdr);
if ( status != 0 )
    return status;
/* Make sure this playback code is compatible with the version of the data in
the stream file. */
if ( ctx->hdr.headerVersion != DS_STREAM_VERSION )
    return kDSVersionErr;
/* Allocate the stream buffers and build the linked list of buffers that are
input to the streamer. */
ctx->bufferList = CreateBufferList(ctx->hdr.streamBuffers,
    ctx->hdr.streamBlockSize);
if ( ctx->bufferList == NULL )
    return kDSNoMemErr;

```

See the *3DO M2 DataStreamer Programmer's Reference* for more information on *FindAndLoadStreamHeader()* and *CreateBufferList()*. Note that you don't have to use *CreateBufferList()*. If you want to allocate the buffers from a different memory allocator or allocate them individually, rather than in one big memory block, you can use *CreateBufferList()* as example code to start from.

Also note that any time the streamer is quiescent (that is, after a stop-flush operation *DSStopStream(..., SOPT_FLUSH)* completes), you can temporarily reuse the stream buffer memory (e.g. for visual transition effects buffers) as long as you don't overwrite the *DSDataBuf* buffer header structures (also allocated by *CreateBufferList()*) which point to them. If *CreateBufferList()* returned a (non-NULL) *firstBp*, then *firstBp->streamData* points to the reusable buffer memory. The buffer memory will be at least *numBuffers * bufferSize* bytes long.

Here's the corresponding cleanup code from *DismantlePlayer()* in the same source code file:

Example 3-2 *Disposing of the buffer list.*

```
FreeMemTrack(ctx->bufferList);
```

Deciding on Stream Block Size

Each buffer that you allocate must be the size of one streamblock. How large this should be depends on what your application does:

- ◆ If you do linear playback and need a high data rate, use large blocks.
- ◆ If you expect lots of branching, use small blocks. This reduces stream bandwidth somewhat, since the stream file contains more filler. However, the streamer will be able to branch more quickly, because the time it takes to branch includes the time required to read the first buffer.

Typical stream block size ranges from 32Kbytes to 96Kbytes. There are typically four buffers.

For More Information

See Chapter 6, "Debugging and Optimization."

Creating a Message Port and Message Items

The application has to create a message port and message items to communicate with the DataStreamer thread. This section first looks at sample code, then discusses using messages in more detail.

Using messages is discussed in detail in Chapter 8 of the *3DO M2 Portfolio Programmer's Guide*.

Example: Setting up a Message Port

Example 3-3 is taken from *InitVideoPlayerFromStreamHeader()* in *playvideostream.c*, which is part of the *VideoPlayer* application that plays back an MPEG stream.

Example 3-4 shows the corresponding cleanup code from *DismantlePlayer()* in *playvideostream.c*.

Example 3-3 *Creating a message port and message items.*

```

/* We need to create a message port and message items to communicate with the
   data streamer. */
ctx->messagePort = NewMsgPort(&ctx->messagePortSignal);
FAIL_NEG("NewMsgPort", ctx->messagePort);

ctx->messageItem = CreateMsgItem(ctx->messagePort);
FAIL_NEG("CreateMsgItem", ctx->messageItem);
ctx->endOfStreamMessageItem = CreateMsgItem(ctx->messagePort);
FAIL_NEG("Create EOS MsgItem", ctx->endOfStreamMessageItem);

```

Example 3-4 *Deleting a message item and message port.*

```

DeleteMsg(ctx->messageItem);
DeleteMsg(ctx->endOfStreamMessageItem);
DeleteMsgPort(ctx->messagePort);

```

The Message-Sending Mechanism

The streamer threads and application threads use messages to communicate with each other. A common message header is used as a preamble to all these messages. The message header is defined in *DataStream.h* as shown in Example 3-5.

Example 3-5 *Message header.*

```

/* The following preamble is used for all types of messages sent by the
   streamer to data acquisition and to subscribers. */
#define DS_MSG_HEADER\
    int32 whatToDo      /* opcode determining msg contents */\
    Item msgItem;       /* message Item for sending this message*/\
    void *privatePtr;   /* ptr to sender's private data */\
    void *link          /* user defined - for linking msgs into lists*/

typedef struct GenericMsg {
    DS_MSG_HEADER;
} GenericMsg, *GenericMsgPtr;

```

Table 3-1 shows the fields in the common message header.

Table 3-1 *Message header fields.*

Field	Description
whatToDo	Specifies the action requested by the message sender. Its value is typically used in a <code>switch ()</code> statement in the receiver, and to discriminate among the union fields in the remainder of the message structure (following the header).
msgItem	The item number of the Portfolio Message Item used to carry this message to the receiver. It is also used to carry the message reply back to the sender.
privatePtr	Stores a pointer to a private data structure needed within the sender.
link	General-purpose linking field to form queues of messages. The thread that currently has this message can use this field to queue messages.

Message Replies

When one of the data stream threads completes a request, it must inform the requester thread via the system routine `ReplyMsg ()`. This routine is required to trigger other actions: For example, when the data acquisition thread replies to a *fill-buffer* request, the reply returns the filled buffer to the streamer, which then adds it to a list of filled buffers ready for distribution to subscriber threads.

When a subscriber thread replies to a *new-data-chunk* message, the reply signals that the subscriber has finished using that data. Once subscribers have signaled that they are finished with all chunks in a buffer, that buffer can be reused.

When a client application sends a *request* message to the streamer, the streamer replies when it has finished processing the request. The client can choose to wait for the reply (synchronous request), or do other work in parallel (asynchronous request). For most request types (*whatToDo*), the streamer will queue additional requests, but will not begin them until the current request is fully processed. The exception to this rule is the *wait-end-off-stream* message. The streamer replies when playback reaches end-of-stream.

Launching the Required Threads

Before a program can start playback, it has to launch the data acquisition, streamer, and subscriber threads. This section looks at some sample code used to launch these threads.

Initializing and Connecting Data Acquisition and Streamer Threads

Initialization of the data acquisition and streamer threads consists of two steps:

1. Call `NewDataAcq()` and `NewDataStream()` to launch the threads:
2. Call `DSConnect()` to connect the streamer and the data acquisition threads.

Example 3-6 is taken from `InitVideoPlayerFromStreamHeader()` in `playvideostream.c`.

Example 3-6 *Initializing data acquisition and streamer threads.*

```
/* Open a data acquisition thread on the specified file */
status = ctx->acqMsgPort =
    NewDataAcq(streamFileName, ctx->hdr.dataAcqDeltaPri);
FAIL_NEG("NewDataAcq", status);

status = NewDataStream(&ctx->streamCBPtr, /*output:stream control block ptr*/
    ctx->bufferList,          /* pointer to buffer list */
    ctx->hdr.streamBlockSize, /* size of each buffer */
    ctx->hdr.streamerDeltaPri, /* streamer thread relative priority */
    ctx->hdr.numSubsMsgs);    /* number of subscriber messages */
FAIL_NEG("NewDataStream", status);

/* Connect the stream to its data supplier */
status = DSConnect(ctx->messageItem, NULL, ctx->streamCBPtr,
    ctx->acqMsgPort);
FAIL_NEG("DSConnect", status);
```

Here's the corresponding cleanup code, taken from `DismantlePlayer()` in `VideoPlayer/playvideostream.c`:

Example 3-7 *Disposing of Data Acquisition and Data Streamer threads*

```
DSConnect(ctx->messageItem, NULL, ctx->streamCBPtr, 0);
DisposeDataAcq(ctx->acqMsgPort);
DisposeDataStream(ctx->messageItem, ctx->streamCBPtr);
```

Initializing and Connecting Subscriber Threads

Example 3-8 shows how to launch and set up subscriber threads. The program that wants to use the DataStreamer has to loop through all subscriber names in the subscriber list found in the header of the stream file. That list is generated by the Weaver from "subscriber" commands in the Weaver script. For each subscriber, the program:

- ◆ Launches the subscriber thread.
- ◆ Performs other subscriber-specific setup as needed.

The following sample code is an example from the VideoPlayer application. This example is taken from *InitVideoPlayerFromStreamHeader()* in *playvideostream.c*. No cleanup code is necessary to dispose of the subscribers, because they will be disposed of automatically when the streamer is disposed of.

Example 3-8 *Launching subscriber threads.*

```
for ( subscriberIndex = 0;
    subsPtr = ctx->hdr.subscriberList + subscriberIndex,
    subsPtr->subscriberType != 0;
    subscriberIndex++ )
{
    switch ( subsPtr->subscriberType )
    {
        case MPVD_CHUNK_TYPE:
            /* Remember the target bit depth */
            ctx->bitDepth = bitDepth;

            /* Instantiate an MPEG video subscriber and sign it up for a
               DataStreamer subscription (now in one easy step!) */
            status = NewMPEGVideoSubscriber(ctx->streamCBPtr, subsPtr->deltaPriority,
                                           ctx->messageItem, bitDepth);
            FAIL_NEG("NewMPEGVideoSubscriber", status);
            ctx->mpegVideoSubMsgPort = status;

            /* ... more MPEG video specific code ... */
            break;

        case SNDS_CHUNK_TYPE:
            status = NewSAudioSubscriber(ctx->streamCBPtr, subsPtr->deltaPriority,
                                         ctx->messageItem);
            FAIL_NEG("NewSAudioSubscriber", status);

            fStreamHasAudio = true;
            break;
    }
}
```

```
case MPAU_CHUNK_TYPE:
    /* ... code for MPEG Audio Subscriber ... */
    break;

case CTRL_CHUNK_TYPE:
    /* the Control Subscriber is obsolete */
    break;

default:
    PERR(("InitVideoPlayerFromStreamHeader - unknown subscriber type
    '%.4s'\n",
        (char*)&subsPtr->subscriberType));
    status = kDSInvalidTypeErr;
    goto FAILED;
}
```

Extra Initialization for the Audio Subscriber

When streaming audio, you need to add certain information to the Weaver script to perform some special initialization not needed for other subscribers:

- ◆ **preloadinstrument**—preloads the DSP instrument that will play back the audio.
- ◆ **enableaudiomask**—specifies a mask that enables specific audio channels at setup.

Example 3-9 shows SAudio audio initialization code from *InitVideoPlayerFromStreamHeader()* in *playvideostream.c*. This is simplified from the example application, which has extra code to handle both SAudio and MPEG audio data types.

Example 3-9 *Initializing the streaming application for SAudio audio.*

```
/* If the stream has audio, then do some additional initializations. */
if ( fStreamHasAudio )
{
    /* Preload audio instrument templates, if any are specified */
    if ( ctx->hdr.preloadInstList != 0 )
    {
        ctlBlock.loadTemplates.tagListPtr = ctx->hdr.preloadInstList;

        status=DSControl(ctx->messageItem, NULL, ctx->streamCBPtr, SNDS_CHUNK_TYPE,
            kSAudioCtlOpLoadTemplates, &ctlBlock);
    }
}
```

```
    FAIL_NEG("DSControl", status);
}

/* Enable any audio channels whose enable bit is set.
 * NOTE: Channel zero is enabled by default, so we don't check it. */
for ( channelNum = 1; channelNum < 32; channelNum++ )
{
    /* If the bit corresponding to the channel number is set, then tell the
     audio subscriber to enable that channel. */
    if ( ctx->hdr.enableAudioChan & (1L << channelNum) )
    {
        status = DSSetChannel(ctx->messageItem, NULL, ctx->streamCBPtr,
            SNDS_CHUNK_TYPE, channelNum, CHAN_ENABLED, CHAN_ENABLED);
        FAIL_NEG("DSSetChannel", status);
    }
}

/* Set the audio clock to use the selected channel */
ctlBlock.clock.channelNumber = ctx->hdr.audioClockChan;
status = DSControl(ctx->messageItem, NULL, ctx->streamCBPtr, SNDS_CHUNK_TYPE,
    kSAudioCtlOpSetClockChan, &ctlBlock);
FAIL_NEG("DSControl - setting audio clock chan", status);
}
```

Registering for End-of-Stream Notification

Before starting playback, one should register for end-of-stream notification. You do this by sending a message to the streamer, as shown in the following example:

Example 3-10 *Registering for end-of-stream notification.*

```
/* Register for end of stream notification. Do it before DSStartStream so
 * we won't miss it in a short or empty stream. */
status = DSWaitEndOfStream(ctx->endOfStreamMessageItem, &EOSMessage,
    ctx->streamCBPtr);
FAIL_NEG("DSWaitEndOfStream", status);
```

The streamer will reply to this message when playback stops, with a status code that indicates why playback stopped. For example, the status `kDSNoErr` indicates a normal end-of-stream, `kDSSTOPChunk` means it reached a STOP chunk and can be resumed, and `kDSAbortErr` means playback aborted due to a fatal error. The status code can also be an I/O error from data acquisition.

Starting the Streaming Process

To start playback, call `DSStartStream()`. It's smart to first preroll the stream, that is, fill most of the stream buffers with data. Example 3-11 shows code from `PlayVideoStream()` in `VideoPlayer/playvideostream.c`:

Example 3-11 *Prerolling and starting stream playback.*

```
PRNT(("Prerolling the stream\n"));
status = DSPreRollStream(ctx->messageItem, NULL, ctx->streamCBPtr, 2);
FAIL_NEG("DSPreRollStream", status);

/* Start the stream running */
PRNT(("Starting the stream\n"));
status = DSStartStream(ctx->messageItem, NULL, ctx->streamCBPtr, 0);
FAIL_NEG("DSStartStream", status);
```

Using Persistent Data With the DATA Subscriber

The DATA subscriber accumulates its data chunks, copies them into a separate buffer, and passes the result to the client. This allows the client to hold onto the data for a long period of time (for example, a background image), without clogging up the stream buffers. The DATA subscriber also allows data to load into memory in a portion of the CD bandwidth. For example, you can load data for the next level of a game while the rest of the stream carries a video introduction/transition to that next level.

Inside Streaming: Control

This chapter looks at adding control to your stream, such as branching in response to control pad input. In that context, it also discusses dealing with time in some detail.

This chapter discusses the following topics:

Topic	Page
Time and the DataStreamer	33
Changing Time within a Stream	34

Time and the DataStreamer

One of the central concepts of streaming is the association of start time and duration with data. The chunks in the stream file have associated time information that is used by the different threads.

- ◆ The weaver multiplexes data into the stream in delivery-time order.
- ◆ Each subscriber watches the stream presentation clock to decide when to deliver its next piece of data.
- ◆ One subscriber (usually a digital audio subscriber) drives the stream presentation clock. All other subscribers follow the clock, enabling audio and video to be synchronized. The stream presentation clock is described below.

The Stream Clock

To manage start time and duration associated with data, the DataStreamer maintains a time base known as the stream presentation clock. "Clock time" is measured in standard Portfolio audio ticks, and runs at a nominal rate of $44100/184 \approx 239.67$ Hz.

To fully understand the details of synchronization in the 3DO M2 DataStreamer, remember that the clock rate is not exactly 240 Hz. However, all chunk-header timestamps within stream files are represented as integers. As a result, the various DataStreamer chunkifier tools must round time to the nearest integer when time-stamping each chunk.

The stream clock can be either running or stopped, but the streamer thread delivers new data to subscribers only if the clock is running.

The presentation time is qualified by a “branch number” in much the same way a CD audio track’s time is qualified by a track number. This means that *<branch number, presentation time>* never goes backwards, even when branching backwards or to another stream.

Note: *If your stream has any time-dependent data, or if you want to do any branching, the subscriber driving the clock must have continuous data.*

Changing Time within a Stream

An interactive stream is almost never played in linear order from beginning to end. Instead, the end-users of your application may be interested in a variety of interactions discussed in this section:

- ◆ Starting and Stopping
- ◆ Jumping to a New Location

Starting and Stopping

To start, stop, or pause a stream, use the functions `DSStartStream()` and `DStopStream()`. If you use `SOPT_FLUSH` as the last argument to either of these two procedures, the streamer and subscribers will flush all of their buffers. If you use `SOPT_NOFLUSH`, the subscriber buffers remain unchanged. If you want to pause and resume, don’t flush when starting and stopping.

Example 4-1 is from *videoplayer/playvideostream.c*, which is part of the VideoPlayer application.

Example 4-1 *Starting and stopping a stream.*

```

if ( jp->startBtn )
{
    /* Pause / Unpause the stream */
    if ( DSIsRunning(ctx->streamCBPtr) )
    {
        /* Note: we don't use the SOPT_FLUSH option when we stop here, so we
           can later resume. */
        status = DSStopStream(ctx->messageItem, NULL, ctx->streamCBPtr,
SOPT_NOFLUSH);
        FAIL_NEG("DSStopStream", status);
    }
    else
    {
        status = DSStartStream(ctx->messageItem, NULL, ctx->streamCBPtr,
SOPT_NOFLUSH);
        FAIL_NEG("DSStartStream", status);
    }
}

```

Using Weaver Script Commands to Place Markers and Stop Chunks

To specify the exact points to which a stream may jump, use the `markertime` and `writemarkertable` Weaver script commands for each point. To specify exact points at which the stream should stop or pause playback, use the `writestopchunk` Weaver script command.

Figure 4-1 shows a Weaver script with commands for placing markers.

Writes header to stream	—	[writestreamheader	
		[streamblocksize	32768
Allocates system resources	—	[streambuffers	4
		[numsubsessages	200
Sets relative thread priorities	—	[streamerdeltapri	-10
		[dataacqdeltapri	-9
Do some setup for audio	—	[preloadinstrument	SA_44K_16_S_CBD2
		[audiolockchan	0
		[enableaudiomask	0x2
Specifies subscribers needed for playback and their relative priorities	—	[subscriber	MPVD -2
		[subscriber	SNDS 11
Specifies file name, priority, and start time	—	[file SF.mpv.chk	1 0
		[file raggae.snds	0 24
Writes a goto chunk, and a stop chunk	—	[writestopchunk	1304
		[markertime	1671
		[writetogotochunk	3351 1 1
		[markertime	3391
Writes a table translating stream time into file offsets	—	[writemarkertable	

Figure 4-1 Weaver script including stream control chunks.

The `markertime` command adds entries to the marker table for the specified times, and the `writemarkertable` command asks the Weaver to write the marker table as a marker table chunk in the output stream.

Note: It's important to place markers at GOP (MPEG Group of Pictures) starts. Examine the output of `MPEGVideoChunkifier` to find out where the GOPs begin.

The `writetogotochunk` command writes a GOTO chunk at the specified time in the output stream (3351 in this example), with the specified branch type (this must be 1) and to the specified marker number (1 in this example).

The `writestopchunk` command writes a STOP chunk at the specified time (in audio clock ticks) in the output stream.

Jumping to a New Location

To branch to a different location, the client application can call `DSGoMarker()`, or put GOTO chunks in the stream.

Using Different Options to DSGoMarker()

Different options to DSGoMarker () allow you to move in your stream in different ways:

- ◆ GOMARKER_ABSOLUTE—Goes to an absolute file byte position. This byte position **must** be the start of a chunk in the stream.
- ◆ GOMARKER_FORWARD and GOMARKER_BACKWARD—Move forward or backward by the specified number of markers relative to the current time's marker.
- ◆ GOMARKER_FORW_TIME, and GOMARKER_BACK_TIME, and GOMARKER_ABS_TIME—Move forward or backward by a specified number of clock ticks or to a specified time.
- ◆ GOMARKER_NUMBER—Goes to the given marker number.

In all cases except GOMARKER_ABSOLUTE, the stream must contain a marker table for the DSGoMarker () command to succeed.

Stream Data Formats

This chapter describes the data format used by the DataStreamer's stream files. The following topics are discussed:

Topic	Page
Some Background on MPEG	39
Some Background on the Data Streamer	41
Stream Control Chunks (STRM)	43
MPEG Video Stream Header Chunk	44
MPEG Video Data Chunks (3 Chunk Subtypes)	44
MPEG Audio Stream Header Chunk	46
MPEG Audio Frame Chunks (3 Current and 9 Potential Subtypes)	47

For More Information

The manpages for 3DO DataStreamer library calls are in Chapter 1, "DataStreamer Functions," of the *3DO M2 DataStreamer Programmer's Reference*.

Data streaming tools are discussed in detail in Chapter 2, "Data Streaming Tools," of the *3DO M2 DataStreamer Programmer's Reference*.

Some Background on MPEG

An elementary stream is a stream of video, audio, or other data type. The following sections describe MPEG audio and video data.

MPEG Audio

In the subset of MPEG Audio that we're interested in, a block of 1152 stereo sample pairs is defined as one *audio presentation unit* or *audio frame*. An audio presentation unit, in compressed form, is called an *audio access unit*. The compression is lossy but not perceptably so.

Each audio access unit is independently coded, so that a decoder can branch to its beginning. Each access unit can also have an associated presentation time-stamp (PTS) for audio-video synchronization (the PTS clock runs at 90 kHz).

The MPEG standard encodes PTSs as unsigned 32-bit values that may wrap around. 3DO's interactive MPEG streamer restricts PTSs to unsigned 32-bit values that *must not* wrap around.

Every audio access unit begins with a 4-byte audio header that gives stream parameters, like sampling rate (44.1 kHz or 48 kHz), and channel mode (mono, stereo, or dual_channel).

MPEG Video

Each video frame is called a *video presentation unit*. In its compressed form, it is called a *video access unit*. As with MPEG audio, the compression is lossy, though artifacts are less noticable at the higher data rates (3.5 Mbit/sec vs. 1.15 Mbit/sec) and when source material is clean of noise. Also, some encoders do a better job than others at deciding which information to discard in order to fit the bandwidth budget.

Each video access unit is either an I-frame, a P-frame, or a B-frame. I-frames are independently coded. They are also the largest unit and the quickest to decode. P-frames are delta-coded from the previous I- or P-frame. B-frames are delta-coded from the previous and next I- or P-frames. B-frames are the smallest unit and take the longest to decode. Video access units are ordered in the stream so that decoding them does not require information later in the stream. This decode order differs from the presentation order (see Figure 5-1).

A typical frame sequence pattern looks like this in presentation order:

B₀ B₁ I₂ B₃ B₄ P₅ B₆ B₇ P₈ B₉ B₁₀ P₁₁ B₁₂ B₁₃ P₁₄

A typical frame sequence pattern looks like this in decode (compressed) order:

I₂ B₀ B₁ P₅ B₃ B₄ P₈ B₆ B₇ P₁₁ B₉ B₁₀ P₁₄ B₁₂ B₁₃

Figure 5-1 *Typical frame sequences*

A video sequence header appears at the start of a video stream and potentially at the start of any group of pictures (GOP). It provides stream parameters, such as picture size, frame rate, and the quantization table.

MPEG video elementary streams contain unambiguous start codes. This makes it possible to start reading at an arbitrary byte in a video stream and find one's place, unlike in MPEG audio.

MPEG Multiplex

MPEG-1 and MPEG-2 together define 3 different formats for multiplexed streams. This is called the *system layer*. The DataStreamer borrows some concepts from the MPEG system layer, but uses a simpler multiplex format.

An MPEG multiplexed stream interleaves packets from its elementary streams. Each packet carries a payload of data bytes from one elementary stream, its time-stamps, and other stream parameters.

There's no information, at the system layer, to indicate where the access units begin within the payload data bytes. For example, an audio packet could contain the last few bytes of one access unit, then two complete access units, then the first bytes of another access unit. This allows optimal linear streaming but causes serious problems for branching, especially given the lack of unambiguous audio start codes.

MPEG video can compress a signal into a target bandwidth. For example, the VideoCD format streams at $75 \text{ blocks/sec} \times 2324 \text{ byte/block} \approx 170 \text{ KByte/sec} \approx 1.33 \text{ Mbit/sec}$. Its video stream uses 1.15 kkbts/sec. Its audio stream uses 224 kbits/sec.

Unlike Cinepak, MPEG sticks to a bandwidth budget. It requires every multiplexer to create streams that won't over/underflow the buffers of an idealized System Target Decoder (STD).

Some Background on the Data Streamer

The "Weaver" is a development tool that multiplexes elementary streams together for the 3DO M2 Data Streamer.

During run-time playback, the Data Streamer thread demultiplexes a woven stream into packets of elementary streams. The Streamer distributes these packets to its subscriber threads, one subscriber per elementary stream type. It distributes references to the packets, not copies of their payloads. Besides compressed audio and video data, data types include texture maps, background data to load while streaming, timed "triggers", and (potentially) executable code.

A subscriber can support multiple channels of its stream type, e.g. for English and French audio. Each channel is a separate elementary stream.

A packet in a woven stream is represented as a *chunk*. Each chunk has a simple header—consisting of an elementary-stream-type code and a 32-bit size field—followed by the chunk data bytes (the payload). This simple format affords a fast parser. Practically all chunks have additional “subscriber common header” fields, containing a presentation timestamp and a channel number.

Every stream file begins with a *stream header chunk* (not to be confused with a chunk header), followed by one header chunk per channel per elementary stream. The stream header chunk supplies initialization parameters to the streamer, and the channel header chunks supply initialization parameters to the subscribers. A HALT chunk can be used to give the subscribers time to initialize (perhaps loading DSP instruments from disc) before playback begins.

The woven stream is blocked into uniform-sized stream blocks, typically 32K, 64K, or 96K. For efficiency and simplicity, all reading is done in units of these blocks, which must be a whole number of disc blocks. Each block contains a sequence of chunks. If needed, a block is filled out with a filler chunk. Chunks may not cross block boundaries.

The streamer uses buffers quite differently than the MPEG STD. The STD demultiplexes by copying data from the mux FIFO buffer into separate FIFOs (one buffer per elementary stream type). The streamer reads the MUX data into stream-block-sized buffers, hands out chunk-references (not copies) to the subscribers, and recycles each buffer when the subscribers are done with all the chunks in it. Empty buffers needn't recycle in FIFO order.

The streamer does its timing—and streams are timestamped—using Audio Folio clock ticks, which run at 44.1 kHz / 184 \approx 239.673913 Hz (assuming the audio sample clock is running at 44.1 kHz).

Markers

Marker points are set during development-time stream processing. They can be created manually at branch points and mechanically at regular time intervals.

A marker refers to a point in the stream—it no longer has to be at the start of a stream block. All data to be presented before that time appears before the marker point. All data to be presented after that time appears after the marker point and must begin independent compression units (meaning that it must be self-contained enough to be decompressed without the previous data).

As the developer, you must decide whether or not to make each marker point begin a new stream block. If a marker point begins a new stream block, it will improve seek covering when branching to that point because the first post-seek block will be full of useful data. The tradeoff is the cost in stream bandwidth and size wasted on a filler chunk.

Subscriber Chunk Common Header Format

The subscriber chunk header format is shown below:

```
typedef struct {
    uint32 chunkType; /* elementary stream data type */
    uint32 chunkSize; /* number of data + header bytes */
    uint32 time; /* DTS/PTS, in audio ticks */
    uint32 channel; /* logical channel number */
    uint32 chunkSubtype; /* data sub-type for this subscriber */
} SubscriberChunkCommon;
```

The `chunkType` field indicates the elementary stream data type (i.e. which subscriber gets this data chunk).

The `time` field is (normally) both a “decode timestamp” (DTS) and “presentation timestamp” (PTS). That is, it tells the stream multiplexer when this data must be delivered to the subscriber for decoding and also tells the subscriber when the decoded data must be presented. Generally, the run-time software should hide any decode delay by offsetting the presentation clock. For MPEG video, this `time` field can only serve as a DTS. A separate PTS is needed in the chunk data.

Note: *The Weaver and Streamer no longer require `chunkSize` to be a multiple of 4. The rule is: each chunk's size tells the number of useful data bytes, and the chunk is followed by 0 to 3 bytes of zeros (not counted in `chunkSize`) to pad it out a mod 4 boundary. This ensures the next chunk will lie on a quadbyte boundary. Every chunkifier that creates non-mod-4 sized chunks must output the pad bytes.*

Notation

```
typedef struct {
    SubscriberChunkCommon common; /* 'MPVD' 'VHDR' */
    ... /* other fields... */
} MPEGAudioHeaderChunk;
```

This chunk format begins with (i.e. it subclasses) `SubscriberChunkCommon`. Its `common.chunkType` field is set to 'MPVD' and its `common.chunkSubtype` field is set to 'VHDR'.

Stream Control Chunks (STRM)

This section describes the 'STRM' chunks, which are the ones the data stream thread processes directly. These chunks control stream playback operations like branching and stopping.

The STRM chunks use the `SubscriberChunkCommon` header, too. The only chunk that doesn't is the FILL chunk, which only has a `chunkType`, a `chunkSize`, and filler bytes.

MPEG Video Stream Header Chunk

An MPEG video stream header chunk gives initialization parameters for the subscriber, plus a data format version number. Every stream with MPEG video must contain one MPEG video stream header chunk per channel (i.e. per elementary stream), before any playback data for that channel.

```
typedef struct {
    SubscriberChunkCommon
        common;          /* 'MPVD', 'VHDR' */
    uint32    version;    /* data format version, currently 1 */
    uint32    maxPictureArea; /* to preallocate ref frames */
    uint32    framePeriod; /* nominal, in 90 kHz ticks */
} MPEGVideoHeaderChunk;
```

The version field gives the data format version for this channel's data chunks. The current version is 1. This field is present for consistency with other subscriber header chunks.

Note: *In practice, if the format changes in the future, we'll probably change the chunkType or chunkSubtype IDs instead of changing the version number.*

The maxPictureArea field gives the channel's maximum picture area in square pixels. It equals horizontalSize * verticalSize, where both factors must be multiples of 8. This information allows the subscriber to preallocate the decoder's reference frame buffers for this channel.

The framePeriod field gives the nominal time between presenting pictures, in units of 90 kHz MPEG ticks. Since every frame in the stream carries its own Presentation Time Stamp (PTS), the framePeriod field is only needed to help the subscriber handle transitions, and perform sanity checking of the PTS values. Note that the video data needn't use an MPEG standard frame rate.

Note: *When encoding a stream, be mindful of the limitations of the current subscriber and decoder implementations. For example, picture width and height must be multiples of 16 pixels.*

MPEG Video Data Chunks (3 Chunk Subtypes)

We store one compressed video frame (access unit) per chunk, but allow it to be split into two chunks to maximize stream bandwidth. This lets the run-time software efficiently process the data as needed (to find the frame's type or single-step). It also lets the stream development tools efficiently compute delivery deadline timestamps (DTSs).

An MPEG video elementary stream gets packetized into one or two `MPEGVideoFrameChunk` chunks per frame (“access unit”). To enable flexible branching, every Group of Pictures is required to begin with a video sequence header.

```
typedef struct {
    SubscriberChunkCommon    common;      /* 'MPVD', 'FRAM' ... */
    uint32                   pts;         /* MPEG PTS, at 90 kHz */
    uint8                    compressedVideo[];
} MPEGVideoFrameChunk;
```

An 'MPVD' 'FRAM' chunk contains an MPEG Presentation Timestamp (PTS) and one frame's bytes of the MPEG video elementary stream. The `common.time` field contains the chunk's delivery time (DTS), in audio ticks.

The `compressedVideo` field of each GOP's *first* 'MPVD' 'FRAM' chunk is:

1. A video sequence header (starting with its `sequence_header_code`)
2. A GOP header (starting with its `group_start_code`),
3. One picture (starting with its `picture_start_code`).

The `compressedVideo` field of every other 'MPVD' 'FRAM' chunk is simply one picture (starting with its `picture_start_code`).

If the frame rate is not an MPEG standard frame rate, put frame rate code 0 (“unknown”) in the video sequence headers.

Restriction: PTSs must not wrap-around within an interactive MPEG stream.

DTSs can be offset from PTSs to compensate for decoder latency in addition to frame reordering.

Quality Boost: The Weaver can greatly reduce stream block fragmentation (increasing usable bandwidth) by splitting an 'MPVD' 'FRAM' chunk (on a 4-byte boundary) into two chunks: an 'MPVD' 'FRM[' chunk which finishes the current stream block and a 'MPVD' 'FRM]' chunk which begins the next stream block. Both chunks have the same DTS (delivery deadline). The Weaver just has to recognize this specific chunkType/Subtype pair, split the chunk, and adjust their chunkSubtype and chunkSize fields. The 'MPVD' 'FRM]' chunk has a `SubscriberChunkCommon` but no `pts` field.

There are no special data alignment or sizing restrictions. For example, an 'MPVD' 'FRAM' or 'MPVD' 'FRM' chunk does not have to be a multiple of 4 bytes long. As always, the chunkifier must follow the chunk with quadbyte padding if needed.

Recommendations

Video streams need not contain any padding strings. A video stream does not need to be constant bit-rate, it just needs to fit within a specific maximum bit-rate.

To conserve stream bandwidth, video sequence headers should not include any sequence_extension_data or user_data, GOP headers should not contain any group_extension_data or user_data, and pictures should not contain any user_data.

Note: *If you want to be able to branch to a specific frame, the best thing to do is to begin a closed GOP there. The streamer can branch to any point in the stream, but the MPEGVideoSubscriber will begin decoding frames at the first frame of the next closed GOP, or the first I frame of the next open GOP.*

As with standard MPEG, if you about two MPEG video elementary streams in the same channel, the latter elementary stream should begin with a closed GOP. Alternatively, it may begin with an open GOP that has the broken link flag turned on, but the decoder will discard all of its frames that preceed (in presentation order) its first I frame.

MPEG Audio Stream Header Chunk

An MPEG audio stream header chunk gives initialization parameters for the subscriber, plus a data format version number. Every stream with MPEG audio must contain one MPEG audio stream header chunk per channel (per elementary stream), before any playback data.

```
typedef struct AudioHeader { /* a std MPEG audio header, 4 bytes */
    unsigned syncword:12,    /* = MPEG_AUDIO_SYNCWORD */
        id:1,                /* = ID_ISO_11172_3_AUDIO */
        layer:2,             /* = audioLayer_II */
    protection_bit:1,        /* 0 => error concealment redundancy */
    bitrate_index:4,         /* index into bitrate table */
    sampling_frequency:2,    /* = 0 => 44.1 KHz */
    padding_bit:1,
    private_bit:1,           /* = 0 (reserved for future use) */
    mode:2,
    mode_extension:2,
    copyright:1,             /* 1 => copyrighted audio stream */
    original_or_copy:1,
    emphasis:2;
} AudioHeader;

typedef struct {
    SubscriberChunkCommon common; /* 'MPAU', 'MHDR' */
    uint32 version; /* data format version = 1 */
    AudioHeader audioHdr; /* an MPEG audio header */
} MPegaAudioHeaderChunk;
```

The parameters are as defined and coded by the MPEG-1 Audio standard:

Layer:	II	[this is a restriction].
sampling_frequency:	44.1 KHz	[this is a restriction].

emphasis: 50/15 μ sec or none.
bit_rate: any Layer II bit rate (384 Kbit/sec is the max).
channel mode: stereo, intensity_stereo, dual_channel, or mono.
private_bit: 0 (reserved for future use).

Restriction: All six parameters must be uniform throughout a channel, except for changes between stereo and intensity_stereo.

When encoding a stream, be mindful of the limitations of the current subscriber and decoder implementations. For example, a subscriber might not treat dual_channel differently than stereo since it has another mechanism to provide multiple channels.

MPEG Audio Frame Chunks (3 Current and 9 Potential Subtypes)

Each audio data chunk contains the bytes of 1 to 4 access units (or “frames”). Since a frame at these settings is \approx 730 bytes, it saves stream bandwidth and CPU time to be able to aggregate a few of them. Putting 3 frames in a chunk reduces the stream bandwidth overhead from 2.7% to 0.9% of the audio data.

```
typedef struct {
    SubscriberChunkCommoncommon; /* 'MPAU', '1FRM' to '4FRM' ... */
    uint8 compressedAudio[];
                                /* 1 to 4 MPEG audio access units */
} MPEGAudioFrameChunk;
```

The common.chunkSubtype field indicates how many audio frames are in the chunk.

Restriction: Each chunk’s compressed audio data must begin and end on byte boundaries. That’s the only data alignment restriction.

As always, the chunkifier must follow the chunk with quadbyte padding if needed.

Quality Boost: The Weaver can reduce stream block fragmentation (increasing usable bandwidth) by splitting an 'MPAU' 'nFRM' chunk (on a 4-byte boundary) into two chunks 'MPAU' 'nFR{' and 'MPAU' 'nFR}' with the same DTS timestamp, for n in [1 .. 4]. For example, 'MPAU' '3FR{' and 'MPAU' '3FR}' chunks make up a 3-frame packet that was split in two.

This common.chunk header is followed by the chunk’s data bytes. The chunkSize field measures the entire chunk size, including header and data bytes.

HALT Chunk

A HALT chunk is used to initialize a subscriber when initialization might disrupt playback by taking a long time or by seeking to another place on the disc that contains the data stream. An example is a chunk that asks the audio subscriber to load DSP instrument files.

A HALT chunk contains a nested subscriber chunk. When the streamer encounters a HALT chunk, it delivers the nested chunk to the appropriate subscriber and then waits (not responding to *any* request messages or other events) until the subscriber finishes processing the nested chunk.

```
typedef struct HaltChunk1 {
    SubscriberChunkCommon common;          /* 'STRM', 'HALT' */
    StreamChunk      subscriberChunk; /* embedded chunk for a
                                         * subscriber */
} HaltChunk1;
```

The subscriberChunk must be properly nested within the HALT chunk, because the HALT chunk's common.chunkSize field measures its own header, plus the nested chunk's header, plus the nested chunk's contents. "Properly nested" means that the subscriberChunk should be embedded in the larger HALT chunk, so that the ending address for both chunks is the same (see Figure 5-2).

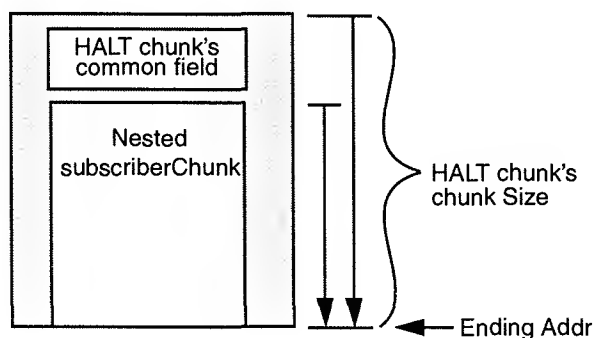


Figure 5-2 Chunk nesting.

GOTO chunk

A GOTO chunk causes a branch in stream playback, as if the client called `DSGoMarker()` at that point during data chunk delivery to subscribers. The GOTO chunk specifies the branch destination and additional options.

```
enum GOTO_Options {
    /* --- Branch type                Meaning of (dest1, dest2) */
    GOTO_OPTIONS_ABSOLUTE    = 0,    /* (byte position, time) */
    /* ... other options ...          */
};
```

```

GOTO_OPTIONS_MARKER      = 1,      /* (marker number, N/A) */
GOTO_OPTIONS_PROGRAMMED  = 2,      /* (programmed-branch number,
                                   N/A) */

/* Bitmask to select the branch type field */
GOTO_OPTIONS_BRANCH_TYPE_MASK= 0xFF,
                                   /* Additional branch option
                                   flags */

GOTO_OPTIONS_FLUSH       = 1<<8   /* flush subscribers */
GOTO_OPTIONS_WAIT        = 1<<9   /* branch then wait for
                                   DSStartStream */};

typedef struct StreamGoToChunk {
    SubscriberChunkCommon common;    /* 'STRM', 'GOTO' */
    uint32 options;                 /* GOTO_Options */
    uint32 dest1;                   /* destination info (depends on
                                   options) */
    uint32 dest2;                   /* destination info (depends on
                                   options) */
} StreamGoToChunk;

```

The options field holds a branch-type code (GOTO_OPTIONS_ABSOLUTE, GOTO_OPTIONS_MARKER, or GOTO_OPTIONS_PROGRAMMED), which determines how the dest1 and dest2 fields express the branch destination, and additional option flags to control how the branch is carried out.

Note: The branch destination must be the start of a stream chunk. It does not have to be the start of a stream block.

Alternative Branch Types

With the GOTO_OPTIONS_ABSOLUTE branch type, the dest1 and dest2 fields give the destination's absolute byte position and presentation time, respectively.

With the GOTO_OPTIONS_MARKER branch type, the dest1 field gives the destination marker number (as for DSGoMarker(..., GOMARKER_NUMBER)), and the dest2 field is unused.

The GOTO_OPTIONS_PROGRAMMED branch type is not implemented.

The “flush” Option

Setting the GOTO_OPTIONS_FLUSH flag in the options field asks the streamer to flush the subscribers when taking a branch.

When branching, the streamer can optionally flush the data that's already queued at subscribers. A “flush” branch gets to the destination as soon as possible, at the cost of freezing the display during the seek. A “non-flush” branch plays out the

pre-branch data before beginning the post-branch data. It uses the pre-branch data to try to cover the seek at the cost of sometimes delaying when the post-branch data begins playing.

When a branch is initiated by a GOTO chunk, the normal case is a non-flush branch that plays out the pre-GOTO data, then plays the post-branch data. When a client calls `DSGoMarker()` it normally requests a flush-branch.

STOP Chunk

A STOP chunk in the stream causes the streamer to stop playback—much as if it hit an early EOF. The streamer will send the end-of-stream notification message with the status `kDSSTOPChunk`.

```
typedef struct StreamStopChunk {
    SubscriberChunkCommon    common;    /* 'STRM', 'STOP' */
    uint32                   options;    /* options [TBD], set to 0 */
} StreamStopChunk;
```

The client can then resume playback by calling `DSStartStream()`.

Note: *The streamer can stop (pause) mid-way into a stream block, so a STOP chunk doesn't have to be followed by a FILL chunk out to the end of the stream block.*

Note: *The STOP chunk has an options field, but no options are yet defined.*

Debugging and Optimization

Most developers find that some fine-tuning of their application yields great gains in playback quality. This section discusses some issues you should consider if your stream drops frames, plays jerkily, or is otherwise not optimal.

Note: *Part of the playback quality does depend on the compression algorithm you are using, the compression parameters, and the quality of the compressor. If you're not satisfied with the quality of your video or audio, consider alternate compression schemes.*

This chapter discusses the following topics:

Topic	Page
Debugging with the DataStreamer Libraries	51
Minimizing Filler	52
Using Appropriate Streamblock Size and Number of Buffers	53
Selecting Thread Priorities	54
Using Tracing to Understand Data Flow	56

Debugging with the DataStreamer Libraries

During development with the DataStreamer Libraries, we strongly recommend that you turn DEBUG on, by compiling these libraries with the `-DDEBUG=1` compile switch. This enables code that checks and reports various error conditions. During development, you need to know if errors arise.

If you choose not to turn on DEBUG, you can still improve the level of error messages and debug printouts by setting `-DPRINT_LEVEL=2` to print all messages, or `-DPRINT_LEVEL=1` to print error messages.

To do source-level debugging on any source files, turn on the `-g` compiler switch and linker switch (preferably in addition to the `-DDEBUG=1` compiler switch), and be sure to turn off the `-Xtrace-table=0` compiler switch.

(The compile-time switch `PRINT_LEVEL` is used by the `<kernel/debug.h>` macros `PERR()`, `PRNT()`, et al.)

Caution: *If printf output occurs during streaming, it may cause a timing glitch. Use the debugger's "Special Mode" to minimize its impact on timing.*

Minimizing Filler

When the Weaver creates a stream file, it uses data chunks that contain no useful information to pad out stream blocks, since each block has to be the same size. These chunks are of type `FILL`, or "filler."

Having filler in the stream essentially wastes data transfer bandwidth. For example, your first attempt at weaving a stream with software-decompressed video might generate 15% filler space. Minimizing filler is important to usable stream bandwidth and seek covering.

Guidelines for Minimizing Filler

The Weaver automatically splits MPEG video chunks, where needed, to fit them into the stream blocks. This largely eliminates filler in each straight-line segment of the stream. In contrast, software-compressed video generates large chunks that can result in large filler chunks.

Optimizing a stream file to minimize filler is an empirical process. You need to experiment with different combinations of parameters to arrive at the best result. Consider yourself successful if the stream contains less than 10 percent filler; less than 6 percent is very good.

Use `dumpstream -stats` to calculate the average filler space. Keep in mind that with a stream header there is usually a lot of filler in the first block, which doesn't affect performance.

Here are some guidelines for minimizing filler:

- ◆ The more markers you place in your stream, the more filler will be present, since the Weaver tool starts a new stream block at each marker.
- ◆ It is often best to adjust the chunking factors on audio or data. Visual data tends to be irregularly sized, whereas audio data tends to have uniformly-sized chunks. Reducing the size of audio or DATA chunks increases the number of chunks present in the stream for a given source file.

- ◆ Having a large number of chunks decreases the efficiency of the stream. Since each chunk contains a bit of header data, the overall percentage of the stream used by the chunk headers goes up.
- ◆ Larger stream blocks make for less fragmentation filler, but result in more marker filler and slower branching.

Using Appropriate Streamblock Size and Number of Buffers

This section discusses how using the right streamblock size and number of buffers can improve performance of your application.

The specific topics in this section are:

- ◆ Streamblock Size Restrictions
- ◆ Dealing with Data Rate Spikes
- ◆ Using More Buffers to Prevent Data Starvation

Streamblock Size Restrictions

You specify streamblock size in the Weaver script using the `streamblocksize` command. When specifying streamblock size, follow these general rules:

- ◆ Streamblock size must be an even multiple of the physical block size of the device. In the case of CD-ROM, the physical block size is 2048 bytes.
- ◆ Typical streamblock values are 32KB to 96 KB (32768–98304 bytes).

Dealing with Data Rate Spikes

The maximum data rate from the CD-ROM is 600 KB/sec (614400 bytes/sec). Because you have to take into account error correction and missing disk revolutions, the average data rate has to be lower. In practice, *brief* periods of data consumption in excess of 600 KB/sec can be achieved with adequate buffer space, as long as the average data consumption rate is 600 KB/sec or less. Figure 6-1 shows how the instantaneous data rate demand (dotted line) fluctuates around the maximum sustained rate (600 KB/sec). While the instantaneous rate is higher than the maximum sustained rate at times, the average is lower and the stream would probably play with no data rate errors.

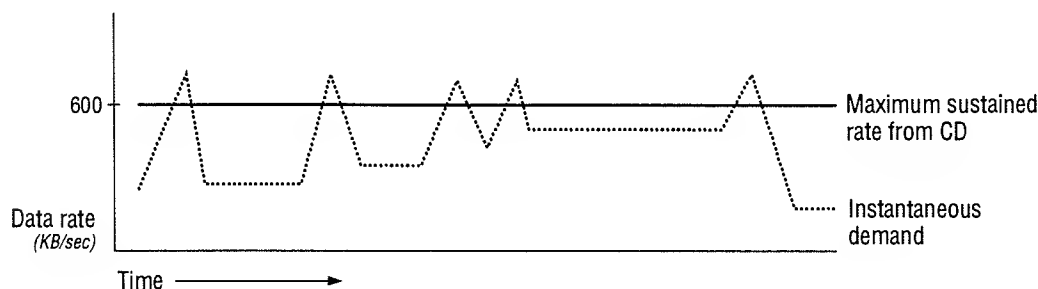


Figure 6-1 Data demand spikes.

Using More Buffers to Prevent Data Starvation

The DataStreamer cannot reuse a stream buffer until it receives a reply for every chunk of data it has sent to subscribers. If the streamer doesn't have enough buffers to continue reading while subscribers are using its data, some of the subscribers may starve for data. To avoid data starvation, consider working with the DATA subscriber. It can copy data out of the stream buffer.

If you allocate a lot of stream buffer space, be sure to allocate enough subscriber messages (using the Weaver script command `numSubsessages`). The Weaver needs enough subscriber messages for the maximum number of chunks in its buffers at one time, plus a few other messages to subscribers (e.g. start/stop, subscribe/unsubscribe). If the streamer runs out of subscriber messages, it will abort stream playback.

Selecting Thread Priorities

Data streaming is built upon the real-time multitasking facilities of the 3DO M2 Portfolio operating system. As in any real-time system, the priorities of the tasks of the application are important. However, the priorities that we are setting here do not determine which process is performed *before* another, they determine which process will *respond the quickest to incoming signals*. With this in mind, remember that **threads that require shorter response latencies get the higher priorities**, and that more buffering relaxes the acceptable response latency.

This section lists a recommended order of process priorities and explains the reasons for that order.

Recommended Order of Process Priorities

Arrange relative process priorities of all DataStreamer threads, including your own program, as follows, starting with the most important:

1. Audio subscriber thread

2. User interface and video display thread (your own application)
3. Video subscriber thread
4. Data Acquisition thread
5. Streamer thread
6. Other subscribers

Keep in mind that this is one recommendation based on certain assumptions about the threads. For one thing, every one of these threads—including the application thread—must process its incoming signals quickly and then go wait for more signals. No polling! Also, the I/O device drivers and folios are operating at a higher priority than all of the streamer threads.

- ◆ The Audio subscriber thread should have the highest priority. This ensures the lowest latency for starting playback of audio buffers and for synchronizing the presentation clock to the timestamps of these started buffers. Even a very tiny gap in audio playback is noticeable. There should be enough stream buffers to keep audio data queued up at the Audio Folio, so a small delay in switching to the audio subscriber thread won't cause an audio glitch.
- ◆ The user interface and video display thread (your application) should have the next highest priority. This minimizes the latency of displaying each frame at its proper time and of noticing user input. This thread must not poll the input device or it will lock out the lower-priority threads.
- ◆ The Video subscriber thread gets the next highest priority, in order to minimize the latency of queueing up the next video frame to the display thread. The buffering between decoding and presentation allows this thread to handle a higher latency than the display thread.
- ◆ The Data Acquisition thread has the next highest priority, in order to minimize the latency of queueing up the next input buffer with the CD-ROM driver. There should be enough stream buffers so that the pacing of the presentation will pace the return of empty buffers for refilling, once the stream gets going.
- ◆ The Streamer thread gets the next highest priority. It spends its time shuffling buffers between the Data Acquisition thread and the subscribers, and the buffer levels should be adequate for it to do this in the background.
- ◆ Other subscriber threads get lower priority, assuming they can handle larger response latencies. This ensures that audio and video play smoothly. However a MIDI audio subscriber should have a priority comparable to the video display thread, because it has about the same time sensitivity.

Note that all priorities discussed above are relative. That is, the actual numeric values of task priorities do not matter as long as the relative priorities are preserved.

How Bad Priorities Can Stop Your Stream

When deciding on subscriber process priorities, consider their effect on the overall utilization of data streaming buffers.

If any subscriber does not reply to its arriving data quickly enough, the stream eventually stops flowing, as in the following situations:

- ◆ If a subscriber owns one data chunk in every stream buffer and doesn't reply to any of them, then the stream stops flowing completely. This can occur if a subscriber has a priority that is so low that it can't operate on its arriving data.
- ◆ A variation of this occurs when a subscriber gets only enough time to process some, but not all, of its data. In this case, buffers containing unprocessed subscriber data chunks remain unavailable for refilling.

This second situation is similar to what happens in the normal case, where the streamer gets a chance to read faster than the subscribers need the data. Once the buffers are full, playback will pace the buffer refilling. This is desirable for covering disc seeks: more data is buffered in memory than is needed right away. While that data is played, a disc seek can occur without causing playback to pause.

No thread—including the application—should be polling. Polling monopolizes the CPU, preventing it from servicing lower-priority threads. Polling also increases the average latency seen by the higher-priority threads, because Portfolio's scheduler can switch more rapidly when it's idle. Every thread should wait for signals, process all received signals, and then loop back to waiting for signals.

Using Tracing to Understand Data Flow

A real-time application like the DataStreamer is difficult to debug with the 3DO M2 Debugger since breakpoints interfere with the running stream. A solution to this problem is to trace events, and the time at which they occur, and write them to a trace log in memory. This makes it possible to examine the event flow without affecting the application.

Tracing acts like a high-level logic analyzer. This functionality is invaluable when analyzing a subscriber's performance, playback smoothness, and thread interactions.

This section explains how to use tracing as part of your application.

How to Use Tracing

Use certain compiler switches to control which of the data streaming threads do tracing. For example, compile the MPEG Video subscriber with these switches:

```
-DDEBUG -DMPVD_TRACE_MAIN=1 -DMPVD_TRACE_SUPPORT=1
```

The various streamer threads (historically just the subscribers) call `AddTrace()` to log each interesting event. `AddTrace()` has very little performance impact; it just stores a few numbers and a time stamp in a trace buffer in memory.

Use the debugger to set the `traceWrap` global variable to control whether `AddTrace()` treats the trace log as a wrap-around buffer or a fill-once buffer. A fill-once buffer is more useful when debugging start-up activities. A wrap-around buffer is more useful when debugging a problem that arises long into playback.

When the test run is finished, `DumpRawTraceBuffer()` gets called to dump the trace log. `DumpTraceCompletionStats()` can also be called to dump <start, completion> event pairs and their durations.

A compile-time switch in `subscribertraceutils.c` controls where the log gets written. It's currently set to printf the log to the 3DODebug terminal. It can be set to write each log to a file in `"/remote"` via the `RawFile` facility, but this hasn't been tested for M2 2.0.

You can add new trace events to the code. Be aware that some of the arguments to `AddTrace()` are used to control event-completion matching in `DumpTraceCompletionStats()`. See the *3DO M2 DataStreamer Programmer's Reference* for more information about `AddTrace()` and `DumpTraceCompletionStats()`.

Run your application as normal, letting it capture trace information into the trace buffer(s) in RAM.

Looking at the Tracing Information

When you run the streamer with tracing on, it writes a trace log(s) to a file in the `/remote` folder.

To look at one of the trace files written to `/remote`, go through these steps:

1. Convert the trace code numbers to mnemonic trace code names. You can do this with the MPW Canon tool, as follows:

```
Canon subscribertrace.dict < tracelog.txt > tracelog.canonized
```

This dictionary is provided in the `SubscriberUtils` directory.

```
--- STREAMER TRACE LOG ---
INDEX    TIME    CHANNEL#    AVALUE    BUFPTR    EVENT
0      144060         0    196938         0    tc3004
1      144060         1    196932         0    tc3004
2      144062    22511    197000         0    tc3002
3      144062    15004    196999         0    tc3002
4      144062    18758    197011         0    tc3002
5      144066         0    197000         0    tc3003
6      144066         0    196999         0    tc3003
7      144066         0    196938         0    tc3005
8      144066    822060      2189    144069    tc3006
9      144066    822060         0         3    tc3007
10     144066         0    196932         0    tc3005
11     144066    15004         40    141920    tc3006
12     144066    15004         0    -2146    tc3007
```

Figure 6-2 *Raw trace example.*

```
--- STREAMER TRACE LOG ---
INDEX    TIME    CHANNEL#    AVALUE    BUFPTR    EVENT
0      144060         0    196938         0    kSubmitMPEGReadBuffer
1      144060         1    196932         0    kSubmitMPEGReadBuffer
2      144062    22511    197000         0    kSubmitMPEGWriteBuffer
3      144062    15004    196999         0    kSubmitMPEGWriteBuffer
4      144062    18758    197011         0    kSubmitMPEGWriteBuffer
5      144066         0    197000         0    kCompleteMPEGWriteBuffer
6      144066         0    196999         0    kCompleteMPEGWriteBuffer
7      144066         0    196938         0    kCompleteMPEGReadBuffer
8      144066    822060      2189    144069    kMPEGReadPTS
9      144066    822060         0         3    kMPEGReadDeltaPTS
10     144066         0    196932         0    kCompleteMPEGReadBuffer
11     144066    15004         40    141920    kMPEGReadPTS
12     144066    15004         0    -2146    kMPEGReadDeltaPTS
```

Figure 6-3 *Trace log after Canon tool has been used.*

2. Open the file in MPW or in a spreadsheet program.

It's also helpful to use the MPW "Search" command (also known as *grep*) to filter the trace log for a specific event. See Figure 6-4.

#	when	ptsValue	audioClockPTS	displayTime	
search -q	kmPEGReadPTS	TraceLog.canonized			
8	144066	822060	2189	144069	kmPEGReadPTS
11	144066	15004	40	141920	kmPEGReadPTS
30	144068	18758	50	144076	kmPEGReadPTS
48	144081	822060	2189	146215	kmPEGReadPTS
54	146220	26265	70	144095	kmPEGReadPTS
60	146224	30019	80	144105	kmPEGReadPTS
66	146228	33772	90	144115	kmPEGReadPTS

Figure 6-4 Using “Search” to filter a trace log.

This trace log reveals incorrect first and fourth PTSs (MPEG Presentation TimeStamps), which explained a presentation hiccup.

To find out what the argument values mean for a particular trace event, “Search” the source code for that trace event code.

Creating New Subscribers

Introduction

Each subscriber thread accepts and processes one type of data from the data stream. The streamer thread sends data to the subscriber threads; your application can then get the data from the subscribers. You may find it useful to create your own subscribers for a special kind of data that isn't handled by available subscribers.

This chapter discusses subscribers with an emphasis on designing your own. Examples of existing subscribers are used to illustrate the technical points. The chapter discusses the following topics:

Topic	Page
Conceptual Background	62
Subscriber Messages	63
Implementing Subscriber Functionality	64
Queued Data and Branching Operations	65

Conceptual Background

In designing a subscriber, it's useful to understand some conceptual background. This section discusses the following topics:

- ◆ Buffering
- ◆ Logical Channels
- ◆ Control Calls

Buffering

Data that a subscriber needs should be delivered before it is needed. The subscriber retains this data until it is used, then returns it to the streamer so that the buffer can be reused. To make the best use of memory for overall streaming throughput, a subscriber should return data to the streamer as soon as possible. Data that is needed for a long time should *never* be left in a stream buffer. Copy the data to long-term storage allocated specifically for that purpose, like the DATA subscriber.

Logical Channels

Logical channels make it possible to weave several independent channels of a given data type into a stream and deal with them independently.

For example, an application could have three logical channels for audio: a sound track, English voice-over, and French voice-over. Users should be able to select their native language, and control music output separately. The client application calls `DSSetChannel()` to tell the subscriber which channels to attend to.

Control Calls

Control over a subscriber is generally implemented by sending messages to the subscriber thread via the streamer thread. The application does this by calling `DSControl()`. Some control messages work for all subscriber types; others are subscriber-specific. Please see the header files and documentation for the subscriber you're interested in.

Subscriber Messages

The streamer thread uses the `SubscriberMsg` structure, defined in *DataStream.h*, to communicate with data subscriber threads. The message header, `DS_MSG_HEADER`, shown in Example 7-1, is common to data acquisition, data streamer, and subscriber messages.

The streamer sends a message to a subscriber whenever it encounters data chunks belonging to that subscriber in the data stream. The subscriber is responsible for doing whatever is appropriate with the data, such as outputting graphics, audio, or other data, and synchronizing it with the stream's presentation clock. As these buffers are used, they must be passed back to the streamer by calling `ReplyMsg()` using the `msgItem` field in the message header. This allows the streamer to reuse the buffer space.

The streamer also sends messages to a subscriber asking it to perform other operations, such as: start and stop playback, get and set status, or a control operation for the client application.

Example 7-1 *SubscriberMsg data structure.*

```

/* --- Opcode values as passed in 'whatToDo' --- */
enum StreamOpcode {
    kStreamOpData      = 0, /* here's a data chunk */
    kStreamOpGetChan = 1, /* get logical channel status */
    kStreamOpSetChan = 2, /* set logical channel status */
    kStreamOpControl = 3, /* perform a subscriber-defined function */
    kStreamOpSync     = 4, /* clock stream resynched the clock */
    kStreamOpStart    = 5, /* stream is being started */
    kStreamOpStop     = 6, /* stream is being stopped */
    kStreamOpOpening  = 7, /* initialize */
    kStreamOpClosing  = 8, /* close-down and exit */
    kStreamOpEOF      = 9, /* data EOF; no more data until branch or switch files */
    kStreamOpAbort    = 10, /* error-case close-down and exit */
    kStreamOpBranch   = 11 /* branch in delivered data (not yet in the clock) */
};

/* --- Messages sent from the streamer to subscribers --- */
typedef struct SubscriberMsg {
    DS_MSG_HEADER;

    union {
        struct {
            /* kStreamOpData */
            void *buffer; /* ptr to the data chunk */
            uint32 branchNumber; /* streamer increments this at each branch */
        } data;
    };
};

```

```
struct {                                /* kStreamOpGetChan, kStreamOpSetChan */
    uint32 number;                      /* channel number to operate on */
    uint32 status;                     /* channel status (bits 31-16 are
                                      subscriber-defined) */
    uint32 mask;                       /* determines which status bits to set */
} channel;

struct {                                /* kStreamOpControl */
    int32 controlArg1;                 /* subscriber-defined */
    void *controlArg2;                 /* subscriber-defined */
} control;

struct {                                /* kStreamOpSync */
    uint32 clock;                     /* current time */
} sync;

struct {                                /* kStreamOpStart */
    uint32 options;                   /* start options (SOPT_NOFLUSH, SOPT_FLUSH)*/
} start;

struct {                                /* kStreamOpStop */
    uint32 options;                   /* stop options (SOPT_NOFLUSH, SOPT_FLUSH)*/
} stop;

struct {                                /* kStreamOpBranch */
    uint32 options;                   /* option:SOPT_NOFLUSH, SOPT_NEWSTREAM*/
    uint32 branchNumber;              /* the new branch number */
} branch;
} msg;

} SubscriberMsg, *SubscriberMsgPtr;
```

Implementing Subscriber Functionality

The functionality of a new subscriber depends on the data you actually want to process. Here are some routines the subscriber code should include:

- ◆ Routine to process arriving data chunks
- ◆ Routine to set the status bits of a given channel
- ◆ Routine to return the status bits of a given channel
- ◆ Routine to perform an arbitrary subscriber-defined control action
- ◆ Routine to close down an open subscription
- ◆ Routine to start all channels for this subscription

- ◆ Routine to stop all channels for this subscription
- ◆ Routine to flush queued data

Subscriber Message Opcodes

Table 7-1 describes the messages the streamer thread may send to a subscriber thread. The subscriber should ignore any messages that do not contain one of the subscriber message opcodes shown in the table's `whatToDo` field.

Table 7-1 *Subscriber message opcodes.*

WhatToDo field	Meaning
<code>kStreamOpData</code>	New data has arrived as specified in <code>msg.data.buffer</code> field. The <code>msg.data.branch</code> number field gives the data's presentation branch number.
<code>kStreamOpGetChan</code>	Request to get logical channel status for the channel specified in <code>msg.channel.number</code> .
<code>kStreamOpSetChan</code>	Request to set logical channel status given the channel number and status specified by <code>msg.channel.number</code> , <code>msg.channel.status</code> , and <code>msg.channel.mask</code> .
<code>kStreamOpControl</code>	Perform a subscriber-defined control function.
<code>kStreamOpSync</code>	Unused
<code>kStreamOpOpening</code>	An initialization message from the streamer that is sent when a subscriber is connected.
<code>kStreamOpClosing</code>	The subscriber should close down and exit.
<code>kStreamOpStart</code>	Start stream playback.
<code>kStreamOpStop</code>	Stop stream playback.
<code>kStreamOpEOF</code>	Unused
<code>kStreamOpAbort</code>	Error; close down and exit.
<code>kStreamOpBranch</code>	The stream branched to a new file, or to a new place in a file. Reset the decoder if needed.

Queued Data and Branching Operations

This section looks at how video and audio data is handled when branching.

Video Data

Consider an interactive fly-through type of stream, where a button press causes a turn down a different path in the fly-through world. There are two options:

- ◆ Graphics from a path that you just departed from should no longer be displayed, so flushing queued data would be appropriate. Note that this branch should take place as fast as possible.
- ◆ When looping through a section of a stream, it might be desirable to use, and not flush, queued data. For example, the loop might be repeated a couple of times, followed by continuing through the stream. This is a non-flush branch—the pre-branch data is not flushed from memory.

Audio Data

Now consider what to do with audio data in the fly-through example. It is undesirable to cease playing the active sound to avoid an audio gap. One way to prevent a gap is to segment the sampled audio or MIDI data in the stream file. As a result, branching from an appropriate departure point to a new section will result in a seamless transition to the audio stored in the stream at the destination of the branching operation. Another way to solve this problem is to use the branching event to trigger a cross fade from the queued audio data to the audio data after the branching operation.

The post-branch data must have data right away for the subscriber driving the presentation clock. Otherwise, when the subscriber finally receives data, it will jump the clock, causing the other subscribers to hiccup their presentations.

Index

Numerics

- 3-2 Pulldown DSG-10
- 3DO DataStreamer
 - data flow overview DSG-21
 - debugging DSG-52
 - examples for use DSG-2
 - future development DSG-2
 - optimizing streaming DSG-52
 - overview DSG-1
 - threads DSG-20
- 3DO SoundHack DSG-12

A

- AddTrace() DSG-57
- AIFC file
 - converting to chunk file DSG-13
- AIFF file
 - converting to chunk file DSG-13
- allocating buffers DSG-22
- analyzing data DSG-7
- animation
 - converting DSG-12
- audio
 - compression DSG-9, DSG-12
 - converting to chunk file DSG-13

- deciding on optimal compression DSG-10

- generating DSG-10

- audio clock ticks DSG-7

- audioclockchan DSG-15

- AudioSubscriber.c DSG-13

B

- branching
 - and buffer size DSG-24
 - and key frames DSG-9

- buffer list
 - creating DSG-22

- buffers
 - allocating DSG-21
 - allocating DSG-9, DSG-22
 - deciding on number DSG-53
 - reuse DSG-26
 - starving buffers DSG-54

C

- channel header chunks DSG-42
- chunk DSG-42
 - and time information DSG-33
 - creating chunk file DSG-13
 - definition DSG-7

chunk types DSG-21
compressedVideo DSG-45
compression
 audio DSG-10
control chunk DSG-6
Control subscriber DSG-17
converting sound DSG-12
CPAK chunk DSG-17
creating
 buffer lists DSG-22
CTRL chunk DSG-7

D

data acquisition thread DSG-20, DSG-21
 initializing DSG-27
data analysis DSG-7
data flow overview DSG-21
data rate
 issues DSG-9
 spikes DSG-53
data rate spikes DSG-9
dataacqdelatpri DSG-15
DATAChunkify DSG-13
DataStreamer *See* 3DO DataStreamer
DEBUG DSG-51
 -DDEBUG=1 DSG-51
 -DPRINT_LEVEL=1 DSG-51
 -DPRINT_LEVEL=2 DSG-51
decode timestamp DSG-43
DSConnect() DSG-27
DSControl() DSG-62
DSGoMarker() DSG-37
 GOMARKER_ABSOLUTE DSG-37
 GOMARKER_BACKWARD DSG-37
 GOMARKER_FORWARD DSG-37
DSStartStream() DSG-34
DSStopStream() DSG-34

dumpstream DSG-9
dumpstream -stats DSG-52

E

enableaudiomask DSG-15, DSG-29
example
 Weaver script DSG-14
examples
 setting up message port DSG-24

F

file DSG-15
filler
 minimizing DSG-52

G

GOTO options DSG-48
 GOTO_OPTIONS_ABSOLUTE
 DSG-49
 GOTO_OPTIONS_FLUSH DSG-49
 GOTO_OPTIONS_MARKER
 DSG-49
Group of Pictures DSG-45

I

I/O bandwidth limitation DSG-8

L

link field DSG-26

M

Macintosh snd resource DSG-12
markertime DSG-35
message header DSG-26
 fields DSG-26
message items DSG-24

message port DSG-24
 example for creating DSG-25
messages DSG-25
MovieCompress DSG-12
MovieEdit DSG-12
MovieToStream DSG-13
MPEG DSG-44, DSG-46, DSG-47
MPEG audio
 audio access unit DSG-40
 audio frame DSG-40
 audio presentation unit DSG-40
MPEG video
 B-frame DSG-40
 I-frame DSG-40
 P-frame DSG-40
 video access unit DSG-40
 video presentation unit DSG-40
MPEGAudioChunkifier DSG-13
MPEGVideoChunkifier DSG-13
msgItem field DSG-26
multitasking DSG-54

N

NewDataAcq() DSG-27
NewDataStream() DSG-27
NTSC hot colors DSG-10
numsubsmessages DSG-15

O

optimizing stream files DSG-52
optimizing streaming DSG-52

P

packets DSG-41
pausing a stream DSG-34
playback
 examples DSG-17

playing a stream DSG-16
PlaySA DSG-17
preloadinstrument DSG-15, DSG-29
preparing a stream file DSG-6
preparing stream files DSG-6
presentation timestamp DSG-43
priorities DSG-54
 problems DSG-56
privatePtr field DSG-26

R

ReplyMsg() DSG-26
resource limitations DSG-8

S

SAudio subscriber DSG-17
SFToStream DSG-13
SNDS chunk DSG-7
Some DSG-39, DSG-41
SOPT_FLUSH DSG-34
SOPT_NOFLUSH DSG-34
sound
 converting DSG-12
SquashSound DSG-12
SSND chunk DSG-17
starting a stream DSG-34
starving buffers DSG-54
stopping a stream DSG-34
Stream DSG-6
stream clock DSG-33
 states DSG-34
stream file
 contents DSG-6
 control chunk DSG-6
 creating DSG-13
 definition DSG-7
 example DSG-6
 header DSG-6

- pausing DSG-34
- playing DSG-16
- preparing DSG-6
- preparing (overview) DSG-6
- starting DSG-34
- stopping DSG-34
- stream header DSG-6
- stream header chunk DSG-42
- stream *See* stream file DSG-7
- stream time DSG-7
- streamblock
 - definition DSG-7
- streamblock size
 - optimizing size DSG-53
- streamblocksize DSG-53
- streamblocksize command DSG-15
- streambuffers DSG-15
- streamer thread DSG-20
 - initializing DSG-27
- streamerdeltapri DSG-15
- subscriber DSG-15
 - connecting DSG-28
 - initializing DSG-28
- subscriber list DSG-17
- subscriber thread DSG-21
- subscriber threads
 - initializing DSG-28
- SubscriberChunkCommon DSG-43
 - common.chunkType field DSG-43
 - compressedVideo field DSG-45
 - framePeriod field DSG-44
 - maxPictureArea field DSG-44
 - version field DSG-44
- SunsubscriberChunkCommon
 - compressedVideo field DSG-45
- system layer DSG-41
- System Target Decoder DSG-41

T

- Telecine process DSG-10
- threads
 - launching DSG-26
 - priorities DSG-54
- Trace
 - AddTrace()
 - traceWrap DSG-57
 - DumpRawTraceBuffer() DSG-57
 - DumpTraceCompletionStats(DSG-57

V

- video
 - compressing DSG-12
 - converting DSG-12
 - generating DSG-10

W

- Weaver DSG-13
 - output DSG-16
 - using DSG-16
- Weaver script DSG-6
 - additions for audio DSG-29
 - example DSG-14
 - starting and stopping streams DSG-35
- whatToDo field DSG-26
- writetogochunk DSG-36
- writemarkertable DSG-35
- writestopchunk DSG-35, DSG-36
- writestreamheader DSG-15



3DO M2 DataStreamer Reference

Version 2.0 – May 1996

Copyright © 1995 The 3DO Company and its licensors.

All rights reserved. This material constitutes confidential and proprietary information of The 3DO Company. This documentation is subject to a license agreement with The 3DO Company and may be used only by parties to such agreement. Use by any other persons, and/or for any purpose not expressly authorized by the agreement, is strictly prohibited.

3DO's LICENSOR(S) MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. 3DO'S LICENSOR(S) DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME JURISDICTIONS. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

Other brand or product names are the trademarks or registered trademarks of their respective holders.

Contents

1 DataStreaming Link Library Calls

Clock

DSClockLE.....	Returns TRUE if (branchNumber1, streamTime1) <= (branchNumber2, streamTime2).	DPR-3
DSClockLT.....	Returns TRUE if (branchNumber1, streamTime1) < (branchNumber2, streamTime2).	DPR-4
DSGetPresentationClock.....	Returns the Data Stream presentation clock and associated values.	DPR-5
DSSetPresentationClock.....	Sets the Data Stream presentation clock and branch number.	DPR-6

DataSubscriber

GetDataChunk.....	Poll for new data on the specified channel.	DPR-8
SetDataMemoryFcns	Set memory allocation/free functions for a DataSubscriber.....	DPR-9

DSBlockFile

--DSBlockFile-Overview--	Overview of the DSBlockFile module.	DPR-10
AsynchReadBlockFile.....	Issues an asynchronous read operation on an open BlockFile.....	DPR-11
CloseBlockFile	Closes a BlockFile.	DPR-13
CreateBlockFileIOReq	Creates an I/O request item for a BlockFile.	DPR-14
GetBlockFileBlockSize	Returns the device block size, in bytes, of an open BlockFile.	DPR-15
GetBlockFileSize	Returns the size, in bytes, of an open BlockFile.....	DPR-16
OpenBlockFile	Opens a BlockFile.	DPR-17
WaitReadDoneBlockFile.....	Waits until an I/O request has completed.	DPR-18

DSUtils

CreateMsgItem	Creates a message item.	DPR-19
NewMsgPort	Creates an unnamed message port.	DPR-20
NewThread	Create a new thread and pass it two arguments.	DPR-21
PollForMsg	Retrieves the next message from the specified message port.	DPR-23

MemPool

--MemPool-Overview--	Overview of the MemPool module.	DPR-24
AllocPoolMem	Allocates an entry from a MemPool.	DPR-25
CreateMemPool	Creates a memory pool (a MemPool).	DPR-26
CreateMemPoolWithOptions	Creates a memory pool (a MemPool) using mem-type options.	DPR-27
DeleteMemPool	Deletes a MemPool.	DPR-28
EnumerateMemPoolEntries	Enumerate all entries in a MemPool.	DPR-29
ForEachFreePoolMember	Enumerate all free entries in a MemPool, e.g. for initialization.	DPR-30
ReturnPoolMem	Returns an entry to a MemPool.	DPR-31

MPEG

FMVCloseDevice	Close an MPEG video device handle.	DPR-32
FMVCreateIOReq	Create an MPEG I/O request setup to signal on I/O completion.	DPR-33
FMVDeleteIOReq	Delete an MPEG I/O request.	DPR-34
FMVGetDeviceItem	Return an FMVDeviceHandle's device item.	DPR-35
FMVGetPTS	Get the PTS data from a completed MPEG video device read request.	DPR-36
FMVOpenVideo	Open a handle to an MPEG video device channel.	DPR-38
FMVReadVideoBuffer	Read one "frame" of decompressed MPEG video data.	DPR-39
FMVSetVideoMode	Set MPEG video device modes.	DPR-40
FMVSetVideoSize	Set the size parameters for an open MPEG video channel.	DPR-41
FMVWriteBuffer	Write a buffer of compressed data to the MPEG video device.	DPR-42

Shutdown

DisposeDataAcq	Shuts down a data acquisition thread.	DPR-44
DisposeDataStream	Shuts down a data stream thread.	DPR-45

Startup

CreateBufferList	Allocates a stream buffer list for use by the streamer.	DPR-46
FillPoolWithMsgItems	Initialize a MemPool of GenericMsg entries.	DPR-48
FindAndLoadStreamHeader	Load a stream file's stream header chunk into memory.	DPR-49
NewDataAcq	Instantiates a new data acquisition thread.	DPR-50
NewDataStream	Instantiates a new data stream thread.	DPR-51
NewDataSubscriber	Instantiates a DATASubscriber.	DPR-53

NewEZFlxSubscriber	Instantiates an EZFlxSubscriber.	DPR-54
NewMPEGAudioSubscriber	Create a new SAudio subscriber thread.	DPR-55
NewMPEGVideoSubscriber	Instantiates an MPEGVideoSubscriber.	DPR-56
NewSAudioSubscriber	Create a new SAudio subscriber thread.	DPR-57

Streaming

DSConnect	Connects/replaces/disconnects a data acquisition client to the stream.	DPR-58
DSControl	Sends a control message to a data stream subscriber. DPR-	DPR-60
DSGetChannel	Read the status of a subscriber's logical channel.	DPR-62
DSGoMarker	Asks the streamer to branch within the stream.	DPR-64
DSIsRunning	Returns TRUE if the Data Stream's presentation clock is currently running.	DPR-67
DSPreRollStream	Start prefilling empty buffers with data.	DPR-68
DSSetChannel	Sets the status of a subscriber's logical stream channel.	DPR-70
DSStartStream	Starts a stream playing.	DPR-72
DSStopStream	Stops stream playback.	DPR-74
DSSubscribe	Add/replace/remove a data stream subscriber.	DPR-76
DSWaitEndOfStream	Register for end-of-stream notification.	DPR-78

Tracing

AddTrace	Add a trace entry to a trace buffer.	DPR-80
DumpRawTraceBuffer	Dump a tabular listing of all entries in a trace buffer.	DPR-82
DumpTraceCompletionStats	Dump stats on <start, completion> event pairs from a trace buffer.	DPR-83

2

DataStreaming Tools

AIFFaudio

AudioChunkifier	Chunkifies an AIFF or AIFC sampled audio file.	DPR-87
SAudioTool	Prints or modifies header information of chunkified AIFF files.	DPR-90

DATA_Subscriber

DATAChunkify	Chunkifies a data file for the DATA subscriber.	DPR-92
--------------------	--	--------

EZFlx

EZFlxChunkifier	Chunkifies an uncompressed QuickTime movie.	DPR-93
QTVideoChunkifier	Chunkifies an EZFlx-compressed QuickTime movie.	DPR-94

MPEG

MPEGAudioChunkifier	Chunkifies an MPEG-1 Audio bitstream file.	DPR-95
MPEGVideoChunkifier	Chunkifies an MPEG-1 Video elementary stream.	DPR-96

Utility

DumpStream	Writes diagnostic listing of a stream.	DPR-98
------------------	---	--------

Weaver

Weaver	Multiplexes chunkified streams into one playable stream.	DPR-99
--------------	---	--------

Weaver_Script_Commands

audioclockchan	Specifies which audio subscriber channel will drive the stream clock.	DPR-101
dataacqdeltapri	Specifies the relative priority for the Data Acquisition thread.	DPR-102
enableaudiomask	Specifies which audio channels to pre-enable.	DPR-103
file	Weaves an input stream file into the woven stream.	DPR-104
markertime	Adds an entry to the marker table to use as a branch destination.	DPR-105
mediablocksize	Specifies the block size of the media. (Use 2048 for CD-ROM.)	DPR-106
numsubsmessages	Specifies number of subscriber messages for the Data Streamer to allocate.	DPR-107
preloadinstrument	Specifies an audio DSP instrument to preload.	DPR-108
streamblocksize	Specifies the stream block size used by the Weaver.	DPR-109
streambuffers	Specifies number of stream buffers the application should allocate.	DPR-110
streamdeltapri	Specifies the relative priority for the Data Stream thread.	DPR-111
streamstarttime	Specifies a time offset for chunks going into the output.	DPR-112
stream. subscriber	Asks to instantiate a subscribe for a particular data type.	DPR-113
writetogotochunk	Writes a STRM GOTO chunk at the specified time in the output stream. ..	DPR-114
writemarkertable	Writes the marker table to the woven output stream.	DPR-116
writestopchunk	Writes a STRM STOP chunk at the specified time in the output stream.	DPR-117
writestreamheader	Writes the stream header to the woven output stream.	DPR-118

Preface

About This Book

This *3DO M2 DataStreamer Reference* contains the man pages for functions and structures used in the 3DO DataStreamer library, and for Weaver script commands.

About the Audience

This document is written for applications developers who need to use the tools, functions and structures listed herein to stream synchronized data to their 3DO M2 system.

How This Book Is Organized

This manual is a command reference. It contains examples of M2 DataStreamer calls, syntax, and use. It contains the following chapters:

Chapter 1, Data Streaming Programmer's Reference—This section presents the reference documentation for the Data Streaming software modules.

Chapter 2, Data Streaming Tools —This section presents the reference documentation for the Data Streaming stream preparation tools.

Related Documentation

The following manuals are useful to developers who are programming in the 3DO environment:

3DO M2 Portfolio Programmer's Guide. A guide to the features of the kernel, IO, filesystem, and so on in the 3DO operating system.

3DO M2 Audio and Music Programmer's Guide. Describes the music and audio folios. It includes overviews, programming tutorials, sample code, and procedure call definitions.

3DO M2 Graphics Programmer's Guide. Describes the function calls for the M2 graphics architecture, folios, and APIs.

3DO M2 Graphics Programmer's Reference. Describes the M2 graphics architecture and APIs. It includes overviews, programming tutorials, sample code, and procedure call definitions.

3DO M2 Audio and Music Programmer's Guide. Describes the music and audio folios. It includes overviews, programming tutorials, sample code, and procedure call definitions.

3DO M2 Debugger Programmer's Guide. A guide to using the 3DO Debugger. It provides a tutorial on using the debugger as well as descriptions of the graphical user interface.

3DO M2 DataStreamer Programmer's Guide. A guide to the 3DO DataStreamer architecture, streaming tools, and weaver tool.

Typographical Conventions

The following typographical conventions are used in this book:

Item	Example
code example	<code>int32 OpenGraphicsFolio(void)</code>
procedure name	<code>CreateScreenGroup()</code>
new term or emphasis	A <i>ViewList</i> is a special case of a View.
file or folder name	The <i>remote</i> folder, the <i>demo.scr</i> file.

Chapter 1

Data Streaming Link Library Calls

This section presents the reference documentation for the Data Streaming software modules.

DSClockLE

Returns TRUE if (branchNumber1, streamTime1) <= (branchNumber2, streamTime2).

Synopsis

```
bool DSClockLE(uint32 branchNumber1, uint32 streamTime1,  
               uint32 branchNumber2, uint32 streamTime2)
```

Description

Compares two DS Presentation Clock (branchNumber, streamTime) pairs and returns TRUE if the first is less than or equal to the second.

The streamer uses a (branchNumber, streamTime) pair to serialize time even when branching backwards or branching to a different stream file.

Arguments

branchNumber1
The first stream presentation clock branch number.

streamTime1
The first stream presentation clock time value.

branchNumber2
The second stream presentation clock branch number.

streamTime2
The second stream presentation clock time value.

Return Value

(branchNumber1, streamTime1) <= (branchNumber2, streamTime2)

Implementation

Streaming library call.

Associated Files

<:streaming:datastreamlib.h>, libds.a

See Also

<:streaming:datastream.h>, DSClockLT(), DSGetPresentationClock(),
DSSetPresentationClock(), DSStartStream(), DSStopStream().

DSClockLT

Returns TRUE if (branchNumber1, streamTime1) < (branchNumber2, streamTime2).

Synopsis

```
bool DSClockLT(uint32 branchNumber1, uint32 streamTime1,  
               uint32 branchNumber2, uint32 streamTime2)
```

Description

Compares two DS Presentation Clock (branchNumber, streamTime) pairs and returns TRUE if the first is less than the second.

The streamer uses a (branchNumber, streamTime) pair to serialize time even when branching backwards or branching to a different stream file.

Arguments

branchNumber1
The first stream presentation clock branch number.

streamTime1
The first stream presentation clock time value.

branchNumber2
The second stream presentation clock branch number.

streamTime2
The second stream presentation clock time value.

Return Value

(branchNumber1, streamTime1) < (branchNumber2, streamTime2)

Implementation

Streaming library call.

Associated Files

<:streaming:datastreamlib.h>, libds.a

See Also

<:streaming:datastream.h>, DSClockLE(), DSGetPresentationClock(),
DSSetPresentationClock(), DSStartStream(), DSStopStream().

DSGetPresentationClock

Returns the Data Stream presentation clock and associated values.

Synopsis

```
void DSGetPresentationClock(DSStreamCBPtr streamCBPtr,  
    DSClock *dsClock)
```

Description

Returns (in *dsClock) the current value of the streamer's presentation clock. The current clock time is calculated as the difference between the current Portfolio audio clock and the streamer's presentation clock offset (or just copied from the streamer's snapshotted value if the stream is stopped).

Presentation time in the DataStream is measured in standard Portfolio audio clock ticks. The DataStream does not change the rate of the audio clock. Since the DataStream does not own the audio clock, however, external tasks can affect the DataStream clock by altering the underlying audio clock.

The DSClock structure contains several values along with the streamTime presentation time itself. There's a bool 'running' indicating if the stream (and thus the clock) is currently running or stopped. There's a branchNumber which allows us to serialize presentation time even when taking a branch backwards in the stream or switching to another stream. <branchNumber, streamTime> (like <trackNumber, time> on a CD) is a monotonically increasing pair. I.e. it never goes backwards. And there's the current audioTime and clockOffset values which (if the stream is running) were used to calculate the streamTime.

This procedure works by accessing the data stream thread's data structures with a semaphore inter-thread lock.

Arguments

streamCBPtr
 Pointer to the stream context block.

dsClock
 Pointer to the DSClock struct to store the results in.

Implementation

Streaming library call.

Associated Files

<:streaming:datastreamlib.h>, libds.a

See Also

<:streaming:datastream.h>, DSSetPresentationClock(), DSStartStream(),
DSStopStream(), DSClockLT(), DSClockLE().

DSSetPresentationClock

Sets the Data Stream presentation clock and branch number.

Synopsis

```
void DSSetPresentationClock(DSStreamCBPtr streamCBPtr,  
    uint32 branchNumber, uint32 streamTime)
```

Description

Sets the Data Stream presentation clock <branchNumber, time>, whether the clock is running or stopped. This is meant to be called by a subscriber that's currently responsible for driving the clock.

*** WARNING: The client application should never call this! ***

Presentation time in the DataStream is measured in standard Portfolio audio clock ticks. The DataStream does not change the rate of the audio clock. Since the DataStream does not own the audio clock, however, external tasks can affect the DataStream clock by altering the state of the audio clock.

The branchNumber allows us to serialize presentation time even when taking a branch backwards in the stream or switching to another stream. <branchNumber, streamTime> (like <trackNumber, time> on a CD) is a monotonically increasing pair. I.e. it never goes backwards.

The streamer increments a delivery-branch-number each time it branches in the stream or switches streams. The streamer includes this number in each data delivery message to a subscriber. The subscriber that's currently driving the clock should use the data's branch number and presentation time to set the presentation clock.

This procedure works by accessing the data stream thread's data structures with a semaphore inter-thread lock.

Caveats

The client application should never call this!

This should ONLY be called by the Data Stream thread and by subscriber threads that adjust the stream clock using their subscription time stamps.

[In the old days, client applications would call DSSetClock() as a hack workaround way to initialize the clock in case there wasn't an audio subscriber or data on an enabled audio channel. Don't do it anymore! The streamer now sets the clock in all cases except no-flush branch, where the audio subscriber must do it.]

Arguments

streamCBPtr

Pointer to the stream context block.

branchNumber

The new presentation branch number (or "sequence number").

streamTime

The new presentation time.

Implementation

Streaming library call.

Associated Files

<:streaming:datastreamlib.h>, libds.a

See Also

<:streaming:datastream.h>, DSSetPresentationClock(), DSStartStream(),
DSStopStream(), DSClockLT(), DSClockLE().

GetDataChunk

Poll for new data on the specified channel.

Synopsis

```
Err GetDataChunk(DataContextPtr ctx, uint32 channelNumber,  
                 DataChunkPtr dataChunkPtr)
```

Description

Check to see if there is a data chunk available on the specified channel, or on every channel if `channelNumber = kEVERY_DATA_CHANNEL`. If `> 0` is returned, the caller is responsible for the block of memory pointed to by `dataChunkPtr->dataPtr`. You should use the data subscriber context function pointer `ctx->freeMemFcn()` to free the memory.

Arguments

`ctx`

The DATA subscriber context (returned by `NewDataSubscriber()`).

`channelNumber`

The number of the channel to check for new data, or `kEVERY_DATA_CHANNEL` to check every channel for available data.

`dataChunkPtr`

A pointer to a `DataChunk` structure which will be filled in if new data is returned.

Return Value

`> 0` if data is returned, `0` if no data returned, negative for an error. Possible error codes currently include:

`kDSSubNotFoundErr`

Invalid subscriber context.

Implementation

Streaming subscriber library call.

Associated Files

`<:streaming:datasubscriber.h>`, `libsubscriber.a`

SetDataMemoryFcns

Set memory allocation/free functions for a DataSubscriber.

Synopsis

```
Err SetDataMemoryFcns(DataContextPtr ctx, DataAllocMemFcn  
allocMemFcn,  
DataFreeMemFcn freeMemFcn)
```

Description

Override the functions used by the DATA subscriber to allocate and free memory (by default, the subscriber calls `AllocMemTrackWithOptions()` and `FreeMemTrack()`). The subscriber makes two allocations for each complete DATA chunk it receives from the stream: one for a header used to track the chunk as its data is accumulated from the stream (chunk type 'DAHD'), and one for the actual data to be copied from the stream (chunk type 'BLOK').

Arguments

`ctx`
The DATA subscriber context (returned by `NewDataSubscriber()`).

`allocMemFcn`
A pointer to the function to call to allocate memory.

`freeMemFcn`
A pointer to the function to call to free memory.

Return Value

Returns zero or a negative error code for failure. Possible error codes currently include:

`kDSSubNotFoundErr`
Invalid subscriber context.

`kDataErrMemsBeenAllocated`
The subscriber has already allocated memory (upon receipt of data from the streamer). This function must be called BEFORE the subscriber receives any stream data.

Implementation

Streaming subscriber library call.

Associated Files

`<:streaming:datasubscriber.h>`, `libsubscriber.a`

See Also

`NewDataSubscriber()`

--DSBlockFile-Overview--

Overview of the DSBlockFile module.

Background

DSBlockFile provides a slightly higher-level API to block-oriented file I/O (higher level than making the system calls directly). Its main advantage is hiding some details of the device's API such as dealing with IOInfo structures and translating byte numbers to block numbers. (But you still have to use byte numbers that are integral multiples of the device's block size. If you want arbitrary byte I/O, use the RawFile procedures.) DSBlockFile is light weight--it doesn't even allocate memory.

DSBlockFile is used by the DataStreamer's DataAcq module.

Usage Overview

Setup: Allocate a BlockFile structure and call `OpenBlockFile()` to fill it in and open the device. Then call `CreateBlockFileIOReq()` one or more times to create IOReq Items. Call `GetBlockFileSize()` to get the file's size and `GetBlockFileBlockSize()` to get the device's block size.

I/O: Call `AsynchReadBlockFile()` to issue each I/O request. This procedure takes byte count and byte offset arguments, but both arguments must be integral multiples of the device's block size. Do not hardwire the block size into your program. For example the "/remote" device has a settable block size.

I/O completion: Use the Item's message or signal completion notification, or `WaitReadDoneBlockFile()` or `WaitIO()`, or `CheckIO()` to tell when each I/O request completes. Don't access the data buffer until the I/O operation has completed. Don't assume that I/O operations complete in the order they were submitted. Look at the completed IOReq Item's `io_Error` field for I/O completion status and its `io_Actual` field for the actual number of bytes read.

Cleanup: Call `DeleteIOReq()` to delete each IOReq Item created for this BlockFile and then call `CloseBlockFile()` to close the device and clean up the BlockFile structure. Or if the thread is about to exit, let the OS automatically reclaim the Device Item and its IOReq Items when the thread exits.

Associated Files

<.:streaming:dsblockfile.h>, libdsutils.a

AsynchReadBlockFile

Issues an asynchronous read operation on an open BlockFile.

Synopsis

```
Err AsynchReadBlockFile(BlockFilePtr bf, Item ioreqItem,  
    void *buffer, uint32 count, uint32 offset)
```

Description

Issues an asynchronous read operation on an open BlockFile. Note that the count and offset are expressed in bytes but **MUST** be multiples of the device's block size.

You can queue up multiple I/O operations using multiple ioReq Items.

You will receive a message or a signal when each I/O request is finished. See `CreateBlockFileIOReq()` for details. You can also check to see if an I/O request is done by calling `CheckIO()`, or wait until it is done by calling `WaitReadDoneBlockFile()` or call `WaitIO()` directly.

Don't access the data buffer until the I/O operation has completed. When the I/O operation completes, look at the IOReq Item's `io_Error` field for I/O completion status, and look at its `io_Actual` field for the actual number of bytes read.

Arguments

bf

A pointer to the open BlockFile structure.

ioreqItem

An IOReq Item created by `CreateBlockFileIOReq()` for this BlockFile.

buffer

The address of the data buffer to read into.

count

The number of bytes to read. This **MUST** be a multiple of the device's block size. (Cf. `GetBlockFileBlockSize()`.)

offset

The offset (or position) into the file to read. This **MUST** be a multiple of the device's block size. (Cf. `GetBlockFileBlockSize()`.)

Return Value

Zero for success, or a negative Portfolio Err code from `SendIO()`.

Assumes

The BlockFile has been opened (via `OpenBlockFile()`).

Implementation

Streaming library call.

Associated Files

<:streaming:dsblockfile.h>, libdsutils.a

See Also

`OpenBlockFile()`, `CreateBlockFileIOReq()`.

CloseBlockFile

Closes a BlockFile.

Synopsis

```
void CloseBlockFile(BlockFilePtr bf)
```

Description

Closes a BlockFile. The client allocates and frees the BlockFile structure. `OpenBlockFile()` fills it in and `CloseBlockFile()` cleans it up.

Before calling `CloseBlockFile()` you should call `DeleteIOReq()` to delete each IOReq Item created for this BlockFile. But if the thread is about to exit, you don't need to do either. The OS will automatically reclaim the Device Item and its IOReq Items when the thread exits.

Arguments

bf
A pointer to the BlockFile structure to close.

Implementation

Streaming library call.

Associated Files

<:streaming:dsblockfile.h>, libdsutils.a

See Also

`OpenBlockFile()`, `AsynchReadBlockFile()`.

CreateBlockFileIOReq

Creates an I/O request Item for a BlockFile.

Synopsis

```
Item CreateBlockFileIOReq(Item deviceItem, Item iodoneReplyPort)
```

Description

Creates an I/O request Item, given the device Item from an open BlockFile. This I/O request Item can be set to give its I/O completion notifications by message or by signal.

Arguments

deviceItem

The open file's device Item from the BlockFile.

iodoneReplyPort

The Item number of a message reply port, or 0 to get I/O completion notification via SIGF_IODONE signal.

Return Value

The new IOReq's Item number, or a negative Err code.

Assumes

The BlockFile has been opened (via `OpenBlockFile()`).

Implementation

Streaming library call.

Associated Files

`<:streaming:dsblockfile.h>`, `libdsutils.a`

See Also

`OpenBlockFile()`, `CloseBlockFile()`, `AsynchReadBlockFile()`.

GetBlockFileBlockSize

Returns the device block size, in bytes, of an open BlockFile.

Synopsis

```
uint32 GetBlockFileBlockSize(BlockFilePtr bf)
```

Description

Returns the device block size, in bytes, of an open BlockFile. You should not hardwire this size into your programs. E.g. the /remote device can be set to various block sizes.

Arguments

bf
A pointer to the open BlockFile structure.

Return Value

The block size, in bytes, of the open BlockFile's device. All I/O operations must be done in integral blocks.

Assumes

The BlockFile has been opened (via `OpenBlockFile()`).

Implementation

Streaming library call.

Associated Files

<:streaming:dsblockfile.h>, libdsutils.a

See Also

`OpenBlockFile()`.

GetBlockFileSize

Returns the size, in bytes, of an open BlockFile.

Synopsis

```
uint32 GetBlockFileSize(BlockFilePtr bf)
```

Description

Returns the size, in bytes, of an open BlockFile.

Arguments

bf
A pointer to the open BlockFile structure.

Return Value

The size, in bytes, of the open BlockFile.

Assumes

The BlockFile has been opened (via `OpenBlockFile()`).

Implementation

Streaming library call.

Associated Files

`<:streaming:dsblockfile.h>`, `libdsutils.a`

See Also

`OpenBlockFile()`.

OpenBlockFile

Opens a BlockFile.

Synopsis

```
Err OpenBlockFile(char *name, BlockFilePtr bf)
```

Description

Opens a BlockFile. The client allocates and frees the BlockFile structure. `OpenBlockFile()` fills it in and `CloseBlockFile()` cleans it up.

Arguments

name

The file name to open.

bf

A pointer to the BlockFile structure to fill in.

Return Value

Zero for success, or a negative Portfolio Err code.

Implementation

Streaming library call.

Associated Files

<.:streaming:dsblockfile.h>, libdsutils.a

See Also

`CloseBlockFile()`, `AsynchReadBlockFile()`.

WaitReadDoneBlockFile

Waits until an I/O request has completed.

Synopsis

```
Err WaitReadDoneBlockFile(Item ioreqItem)
```

Description

Waits until an I/O request (issued by `AsynchReadBlockFile()` on an open `BlockFile`) has completed. This simply calls `WaitIO()`, which you could call directly.

Arguments

`ioreqItem`
The `IOReq` Item.

Return Value

Zero for success, or a negative Portfolio Err code from `WaitIO()`.

Implementation

Streaming library call.

Associated Files

`<:streaming:dsblockfile.h>`, `libdsutils.a`

See Also

`OpenBlockFile()`, `CreateBlockFileIOReq()`, `AsynchReadBlockFile()`.

CreateMsgItem

Creates a message Item.

Synopsis

```
Item CreateMsgItem(Item replyPort)
```

Description

This function creates an unnamed message Item with the default priority. It's used by the data streaming modules as a convenience function for `CreateMsg()`.

You can provide an optional reply port. Any task you send the message to will be able to reply it to you by calling `ReplyMsg()`. If you don't supply a reply port (i.e. if `replyPort == 0`) then the act of sending the message to another task will also transfer the ownership of the message to the receiving task.

Arguments

`replyPort`

The Item number of the message port at which to receive the reply, or 0 for no reply port.

Return Value

The item number of the message or a negative error code for failure.

Implementation

Streaming library call.

Associated Files

`<:streaming:msgutils.h>`, `libds.a`

See Also

`CreateMsg()`, `DeleteMsg()`

NewMsgPort

Creates an unnamed message port.

Synopsis

```
Item NewMsgPort(uint32 *signalMaskPtr)
```

Description

`NewMsgPort()` creates a message port and a signal bit for it. This returns the message port's Item number and optionally also returns its signal mask by storing into `*signalMaskPtr`. This is a convenience routine used in the streaming library.

Call `DeleteMsgPort()` to delete the message port Item and its associated signal bit.

Arguments

`signalMaskPtr`

Address to store the new message port's signal mask, or NULL if you don't need this information.

Return Value

Message port Item number or a negative error code.

Implementation

Streaming library call.

Associated Files

`<:streaming:msgutils.h>`, `libds.a`

See Also

`CreateMsgPort()`, `DeleteMsgPort()`

NewThread

Create a new thread and pass it two arguments.

Synopsis

```
Item NewThread(void *threadProcPtr,
               int32 stackSize,
               int32 threadPriority,
               char *threadName,
               int32 argc,
               void *argp)
```

Description

Create a new thread given the proc pointer, stack size, thread priority, and thread name, and two arguments to pass to it (known as argc and argv by convention).

This just calls `CreateThreadVA()` and supplies tag args to pass two arguments to the new thread. `NewThread()` was much more useful before `CreateThreadVA()` existed.

To dispose the thread call `DisposeThread()` (which is just a macro for `DeleteThread()`). Or have the thread call `exit()` or simply return. In any case, the OS will automatically deallocate the thread's stack.

Arguments

`threadProcPtr`

The new thread begins with a call to this procedure. `threadProcPtr` should really be declared `(*threadProcPtr)(int32 argc, void *argp)`, but the caller would still end up type casting due to `argc/argp`.

`stackSize`

Size of the stack space to allocate for the new thread. `NewThread()` will round it up to a mod 8 value and allocate it mod 8 aligned. If there isn't enough memory to allocate it, `NewThread()` will return `NOMEM`.

`threadPriority`

Priority for the new thread.

`threadName`

A name (a C character string) for the new thread. If `NULL`, `NewThread()` will return `BADNAME`.

`argc`

The first arg to pass to `threadProcPtr()`. It's known as `argc` and declared `int32` per the C model `main(argc, argv)`, but it can be any 32-bit value that `threadProcPtr()` wants.

`argp`

The second arg to pass to `threadProcPtr()`. It's known as `argp` and declared `void*` per the C model `main(argc, argv)`, but it can be any 32-bit value that `threadProcPtr()` wants.

Return Value

The item number of the new thread `Item`, or an error code such as `NOMEM` (couldn't allocate the stack) or `BADNAME` (the `threadName` can't be `NULL`).

Implementation

Streaming library call.

Associated Files

`<:streaming:threadhelper.h>`, `libdsutils.a`

See Also

`DisposeThread()`, `DeleteThread()`, `CreateItem()`, `CreateThread()`.

PollForMsg

Retrieves the next message from the specified message port.

Synopsis

```
bool PollForMsg(Item msgPortItem, Item *msgItemPtr, Message*  
*pMsgPtr,  
void* *pMsgDataPtr, Err *status)
```

Description

PollForMsg() calls GetMsg() to retrieve the next message waiting at the given message port, and returns its Message address and Message data pointer. PollForMsg() returns immediately if there is no message waiting.

PollForMsg() is just a convenience routine. One could directly call msgItem = GetMsg(msgPortItem) and if (msgItem > 0), get MESSAGE(msgItem)->msg_DataPtr.

Arguments

msgPortItem

Item number of the message port to look for messages.

msgItemPtr

Address to store the received message's Item number, if any. Pass in NULL if you don't need this information.

pMsgPtr

Address to store the received message's Message address. Pass in NULL if you don't need this information.

pMsgDataPtr

Address to store the received message's data address. Pass in NULL if you don't need this information.

status

Address to store the resulting status Portfolio error code. Pass in NULL if you don't need this information.

Return Value

TRUE if a message was successfully received. FALSE if there was no message waiting or an error occurred.

Implementation

Streaming library call.

Associated Files

<:streaming:msgutils.h>, libds.a

See Also

GetMsg()

--MemPool-Overview--

Overview of the MemPool module.

Background

A MemPool holds a fixed number of fixed-sized pool entries (aka "nodes" or "members"). Entries can then be repeatedly allocated from the pool via `AllocPoolMem()` and returned to the pool via `ReturnPoolMem()` with just a few instructions and with no risk of memory fragmentation.

Typically, the client will initialize the pool entries to save work for each reuse of the entries. MemPool will not alter the contents of free pool entries.

MemPools are handy for data streaming and other real-time processing.

Usage Overview

Call `CreateMemPool()` or `CreateMemPoolWithOptions()` to create (allocate) a MemPool. The latter lets you supply `AllocMem()` options such as (`MEMTYPE_DMA | MEMTYPE_FILL`).

If you allocate with `MEMTYPE_FILL`, the pool memory will be zeroed out. To further initialize the pool entries, pass an entry-initialization function to `ForEachFreePoolMember()`. E.g. for a pool of I/O-tracker entries, your entry-initialization function could create an `IOReq` Item, call `LookupItem()` and save the result, and setup `IOInfo` fields.

Call `AllocPoolMem()` to allocate a node from a MemPool, and call `ReturnPoolMem()` to return the node to the MemPool.

Call `EnumerateMemPoolEntries()` to enumerate ALL nodes in a MemPool. This is mainly intended for debugging, e.g. to locate a forgotten pool entry.

Call `DeleteMemPool()` to delete (deallocate) a MemPool.

Associated Files

<:streaming:mempool.h>, libdsutils.a

AllocPoolMem

Allocates an entry from a MemPool.

Synopsis

```
void *AllocPoolMem(MemPoolPtr memPool)
```

Description

Allocates an entry (or "node" or "member") from a MemPool. This is implemented as a very fast singly-linked list operation.

Arguments

memPool
The MemPool to allocate from.

Return Value

A pointer to the entry data allocated from the MemPool. If there are no free entries, this returns NULL.

Implementation

Streaming library call.

Associated Files

<:streaming:mempool.h>, libdsutils.a

See Also

CreateMemPool(), ReturnPoolMem()

CreateMemPool

Creates a memory pool (a MemPool).

Synopsis

```
MemPoolPtr CreateMemPool(int32 numToPreallocate, int32 sizeofEntry)
```

Description

Creates (allocates) a MemPool by calling CreateMemPoolWithOptions(numToPreallocate, sizeofEntry, MEMTYPE_FILL), thus allocating a pool of zeroed-out entries.

Call ForEachFreePoolMember() to further initialize the pool entries.

Arguments

numToPreallocate

Number of entries in the new MemPool.

sizeofEntry

Size of each pool entry, in bytes. The size gets rounded up to the nearest multiple of 4 bytes so each pool entry will start on a word boundary.

Return Value

A pointer to the new MemPool, or NULL if CreateMemPool failed.

Implementation

Streaming library call.

Associated Files

<:streaming:mempool.h>, libdsutils.a

See Also

CreateMemPoolWithOptions(), DeleteMemPool(), AllocPoolMem(),
ReturnPoolMem(), ForEachFreePoolMember()

CreateMemPoolWithOptions

Creates a memory pool (a MemPool) using mem-type options.

Synopsis

```
MemPoolPtr CreateMemPoolWithOptions(int32 numToPreallocate,  
                                     int32 sizeofEntry, uint32 memtypeOptions)
```

Description

Creates (allocates) a MemPool. You can supply AllocMem() options such as (MEMTYPE_DMA | MEMTYPE_FILL).

MemPools support data streaming and other real-time processing.

If you allocate with MEMTYPE_FILL, the pool memory will be zeroed out. To further initialize the pool entries, pass an entry-initialization function to ForEachFreePoolMember(). E.g. for a pool of I/O-tracker entries, your entry-initialization function could create an IOReq Item, call LookupItem() and save the result, and setup IOInfo fields.

Arguments

numToPreallocate

Number of entries to create in the new MemPool.

sizeofEntry

Size to create each pool entry, in bytes. Each pool entry will start on a mod-4 byte boundary.

memtypeOptions

AllocMem() options such as (MEMTYPE_DMA | MEMTYPE_FILL).

Return Value

A pointer to the new MemPool, or NULL if CreateMemPoolWithOptions failed.

Implementation

Streaming library call.

Associated Files

<:streaming:mempool.h>, libdsutils.a

See Also

CreateMemPool(), DeleteMemPool(), AllocPoolMem(), ReturnPoolMem(),
ForEachFreePoolMember(), AllocMem()

DeleteMemPool

Deletes a MemPool.

Synopsis

```
void DeleteMemPool(MemPoolPtr memPool)
```

Description

Deletes (deallocates) the specified MemPool by calling `FreeMem()`.

Arguments

`memPool`
The MemPool to delete.

Implementation

Streaming library call.

Associated Files

`<:streaming:mempool.h>`, `libdsutils.a`

See Also

`CreateMemPool()`, `FreeMem()`

EnumerateMemPoolEntries

Enumerate all entries in a MemPool.

Synopsis

```
bool EnumerateMemPoolEntries(MemPoolPtr pool,  
    ForEachPoolMemberFuncPtr forEachFunc, void *argValue)
```

Description

Applies the function `forEachFunc` to each entry in a MemPool. For each entry in the MemPool, this passes the input `argValue`, the entry data pointer, and an in-use `bool` to `forEachFunc`.

This is mainly intended for debugging, e.g. to locate a forgotten pool entry.

Arguments

`pool`
MemPool to scan.

`forEachFunc`
Function to apply to each entry in the pool: `forEachFunc(void *argValue, void *poolEntry)`. If `forEachFunc()` returns `FALSE`, `EnumerateMemPoolEntries()` will immediately stop scanning the pool and will return `FALSE`.

`argValue`
Argument string passed to the function.

Return Value

If `forEachFunc()` ever returns `FALSE`, `EnumerateMemPoolEntries()` will immediately return `FALSE`. Otherwise `EnumerateMemPoolEntries()` will return `TRUE`.

Implementation

Streaming library call.

Associated Files

<:streaming:mempool.h>, libdsutils.a

See Also

`CreateMemPool()`, `ForEachFreePoolMember()`

ForEachFreePoolMember

Enumerate all free entries in a MemPool, e.g. for initialization.

Synopsis

```
bool ForEachFreePoolMember(MemPoolPtr memPool,  
    ForEachFreePoolMemberFuncPtr forEachFunc, void *argValue)
```

Description

Applies the function `forEachFunc` to each free entry in a MemPool. For each free entry in the MemPool, this passes the input `argValue` and the entry data pointer to `forEachFunc`.

This is handy for initializing the entries in a new pool and for cleaning up the entries in a pool about to be deleted.

Arguments

`memPool`
The MemPool to scan.

`forEachFunc`
Function to apply to each free entry in the pool: `forEachFunc(void *argValue, void *poolEntry)`. If `forEachFunc()` returns FALSE, `ForEachFreePoolMember()` will immediately stop scanning the pool and will return FALSE.

`argValue`
Client data argument to pass through to `forEachFunc`. This lets you supply context info to `forEachFunc()`.

Return Value

If `forEachFunc()` ever returns FALSE, `ForEachFreePoolMember()` will immediately return FALSE. Otherwise `ForEachFreePoolMember()` will return TRUE. (If quitting the scan early leaves the pool entries in an incomplete state, it's up to the caller to clean up any side effects.)

Implementation

Streaming library call.

Associated Files

<:streaming:mempool.h>, libdsutils.a

See Also

`CreateMemPool()`, `EnumerateMemPoolEntries()`

ReturnPoolMem

Returns an entry to a MemPool.

Synopsis

```
void ReturnPoolMem(MemPoolPtr memPool, void* poolEntry)
```

Description

Returns an entry to a MemPool, i.e. "deallocates" the entry back to the pool. This is implemented as a very fast singly-linked list operation. ASSUMES: memPool is the MemPool that poolEntry was allocated from.

Arguments

memPool

The MemPool to return poolEntry to.

poolEntry

The MemPool entry to return to memPool.

Implementation

Streaming library call.

Associated Files

<:streaming:mempool.h>, libdsutils.a

See Also

AllocPoolMem(), CreateMemPool()

FMVCloseDevice

Close an MPEG video device handle.

Synopsis

```
Err FMVCloseDevice(FMVDeviceHandle *device)
```

Description

Close an MPEG video device handle. It's ok to call this if the device wasn't successfully opened.

Arguments

device

A pointer to an FMVDeviceHandle to close.

Return Value

kDSBadPtrErr (defined in dserror.h) if the device argument was NULL, else the Portfolio Err code from the device `CloseItem()` call.

Implementation

Link library call implemented in libsubscriber.a.

Associated Files

"fmvdriverinterface.h", libsubscriber.a

See Also

`FMVOpenVideo()`.

FMVCreateIOReq

Create an MPEG I/O request setup to signal on I/O completion.

Synopsis

```
Item FMVCreateIOReq(FMVDeviceHandle *device, uint32
completionSignal)
```

Description

This function creates an MPEG device I/O request Item. The Item will send you a signal on I/O completion, either completionSignal or SIGF_IODONE (if completionSignal is 0).

This is a low-overhead notification mechanism. It doesn't tell you which I/O request(s) completed, but it doesn't matter since you should process I/O completions in request order. That is, keep a queue of your outstanding I/O requests, and when the signal arrives, process any and all completed I/O requests at the head of the queue.

Arguments

device

A pointer to an FMVDeviceHandle.

completionSignal

The completion signal mask (a 1-bit mask identifying a signal number) or 0 (to use the pre-allocated signal SIGF_IODONE).

Return Value

The Item number of the new I/O request or one of the following negative error codes if an error occurs:

kDSBadPtrErr (defined in dserror.h)

The device argument was NULL.

ER_Kr_ItemNotOpen

The device specified by the dev argument is not open.

NOMEM

There was not enough memory to complete the operation.

Implementation

Link library call implemented in libsubscriber.a.

Associated Files

"fmvdriverinterface.h", libsubscriber.a

Notes

When you no longer need an I/O request, use FMVDeleteIOReq() to delete it.

See Also

FMVOpenVideo(), FMVDeleteIOReq(), FMVWriteBuffer(),
FMVReadVideoBuffer().

FMVDeleteIOReq

Delete an MPEG I/O request.

Synopsis

```
Err FMVDeleteIOReq(Item item)
```

Description

This macro deletes an MPEG device I/O request `Item` created by `FMVCreateIOReq()`.

Arguments

`item`
The IOReq Item created by `FMVCreateIOReq()`.

Return Value

0 or a negative error code.

Implementation

Macro call implemented in "fmvdriverinterface.h".

Associated Files

"fmvdriverinterface.h", libsubscriber.a

See Also

`FMVOpenVideo()`, `FMVCreateIOReq()`.

FMVGetDeviceItem

Return an FMVDeviceHandle's device Item.

Synopsis

```
Item FMVGetDeviceItem(FMVDeviceHandle *anFMVDeviceHandlePtr)
```

Description

Return an FMVDeviceHandle's device Item. Use this in case you need to operate directly on the Item. (Normally the `fmvdriverinterface.h` routines will suffice and you won't need to operate on the Item directly.)

Arguments

`anFMVDeviceHandlePtr`
A pointer to an FMVDeviceHandle structure.

Implementation

Macro call implemented in "fmvdriverinterface.h".

Associated Files

"fmvdriverinterface.h", libsubscriber.a

See Also

`FMVOpenVideo()`.

FMVGetPTS

Get the PTS data from a completed MPEG video device read request.

Synopsis

```
Err FMVGetPTS(IOReq *ioReqItemPtr, uint32 *ptsValue, uint32
*userData)
```

Description

A "PTS" is an MPEG Presentation Time Stamp. It tells when to present the decoded video data. It's mainly useful to synchronize the video and audio presentation.

A PTS value is in 90 kHz MPEG ticks. The MPEG device returns the low-order 32 bits of PTS, although MPEG defines it as a 33-bit field.

A user data value can be passed through the decoder along with the PTS value. For instance the Data Streamer uses it for a "branch number" value, so <branchNumber, PTS> is monotonically increasing even if the program jumps backwards in the stream or switches streams.

FMVGetPTS() gets the PTS data from a completed MPEG video device read request. You supply storage for the results.

Arguments

ioReqItemPtr

ptr to the looked-up IOReq of a completed MPEG device read request.

ptsValue

Address to store the PTS value into.

userData

Address to store the user data value into.

Return Value

0 if successful.

kDSMPEGDevPTSNotValidErr if this read request didn't emit a valid PTS.

kDSMPEGDevPTSNotValidErr is an info value, not an error condition. Once the MPEG video decoder has a valid input PTS, it will emit a valid PTS on every decoded frame for the rest of the contiguous sequence.

In *ptsValue, the low-order 32-bits of the PTS value, or 0 if the read request has no valid PTS value.

In *userData, the user data value passed through the decoder along with the PTS value (or unchanged if the read request has no valid PTS value). Cf. FMVWriteBuffer().

Implementation

Link library call implemented in libsubscriber.a.

Associated Files

"fmvdriverinterface.h", libsubscriber.a

See Also

FMVOpenVideo(), FMVWriteBuffer(), FMVReadVideoBuffer().

FMVOpenVideo

Open a handle to an MPEG video device channel.

Synopsis

```
Err FMVOpenVideo(FMVDeviceHandle *device, int32 bitsPerPixel,  
                int32 horizSize, int32 vertSize, uint8 resampleMode)
```

Description

The module "fmvdriverinterface" provides a slightly higher level interface to the MPEG device driver. It also provides some isolation from change, e.g. the rest of the streaming software didn't have to change when the device was renamed from the "FMV" device to the "MPEG" device.

This function opens and initializes a handle to an MPEG driver video channel.

To close the FMVDeviceHandle, call FMVCloseDevice().

See FMVSetVideoSize() for more documentation on the input parameters.

Arguments

device

FMVDeviceHandle* to fill in; it should already be initialized.

bitsPerPixel

16 or 24.

horizSize

Typically 320 (NTSC) or 384 (PAL).

vertSize

Typically 240 (NTSC) or 288 (PAL).

resampleMode

kCODEC_SQUARE_RESAMPLE noop resampling on M2 or Opera,

kCODEC_NTSC_RESAMPLE 352 -> 320 resampling on Opera only, or

kCODEC_PAL_RESAMPLE 352 -> 384 resampling on Opera only.

These constants are defined in *<:device:mpegvideo.h>*.

Return Value

0 if successful, a negative error code if unsuccessful.

Implementation

Link library call implemented in libsubscriber.a.

Associated Files

"fmvdriverinterface.h", libsubscriber.a

See Also

FMVCloseDevice().

FMVReadVideoBuffer

Read one "frame" of decompressed MPEG video data.

Synopsis

```
Err FMVReadVideoBuffer(void *buffer, int32 numBytes, Item ioReqItem)
```

Description

This issues an asynchronous I/O request to read one "frame" of decompressed MPEG video data from the MPEG device into the supplied buffer. After the I/O completes, you can call `FMVGetPTS()` to get the data's PTS (MPEG Presentation Timestamp).

Each read request returns a separate video frame. If the buffer isn't large enough, the driver will return part of the frame.

Arguments

buffer

Address of the data buffer to read decompressed MPEG data into.

numBytes

number of data bytes in the buffer, must be a multiple of 2.

ioReqItem

MPEG IOReq Item from `FMVCreateIOReq()`.

Return Value

Error code from `SendIO()`.

Notes

The MPEG devices has buffer alignment and size modulo requirements. Also, each buffer better be large enough to hold an entire frame, where bytes/pixel is 2 in 16 bit/pixel mode and 4 in 24 bit/pixel mode.

Implementation

Link library call implemented in `libsubscriber.a`.

Associated Files

"`fmvdriverinterface.h`", `libsubscriber.a`

See Also

`FMVOpenVideo()`, `FMVCreateIOReq()`.

FMVSetVideoMode

Set MPEG video device modes.

Synopsis

```
Err FMVSetVideoMode(FMVDeviceHandle *device, int32 mode,
                    int32 argument)
```

Description

Set MPEG video device modes.

Arguments

device

An open video FMVDeviceHandle*.

mode

VID_CODEC_TAG_PLAY to cancel decode-only-I-frames mode, or

VID_CODEC_TAG_SKIPFRAMES to skip <argument> number of B frames, more precisely, to add <argument> to the skip counter (there's no way to clear this counter short of FMVCloseVideo), or

VID_CODEC_TAG_KEYFRAMES to go into decode-only-I-frames mode, or

etc.

argument

The argument to the mode being set, e.g. the number of additional B frames to skip if mode == VID_CODEC_TAG_SKIPFRAMES.

Return Value

Error code from DoIO().

Assumes

The channel was opened via FMVOpenVideo().

Implementation

Link library call implemented in libsubscriber.a.

Associated Files

"fmvdriverinterface.h", libsubscriber.a

See Also

FMVOpenVideo(), FMVCreateIOReq().

FMVSetVideoSize

Set the size parameters for an open MPEG video channel.

Synopsis

```
Err FMVSetVideoSize(FMVDeviceHandle *device, int32 bitsPerPixel,  
                    int32 horizSize, int32 vertSize, uint8 resampleMode)
```

Description

Set the size parameters for an open MPEG video channel.

Arguments

device
An open MPEG video FMVDeviceHandle*.

bitsPerPixel
16 or 24.

horizSize
Typically 320 (NTSC) or 384 (PAL).

vertSize
Typically 240 (NTSC) or 288 (PAL).

resampleMode
kCODEC_SQUARE_RESAMPLE noop resampling on M2 or Opera,
kCODEC_NTSC_RESAMPLE 352->320 resampling on Opera only, or
kCODEC_PAL_RESAMPLE 352->384 resampling on Opera only.

These constants are defined in `<:device:mpegvideo.h>`.

Return Value

Error code from `DoIO()`.

Assumes

The channel was opened via `FMVOpenVideo()`.

Implementation

Link library call implemented in `libsubscriber.a`.

Associated Files

"`fmvdriverinterface.h`", `libsubscriber.a`

Notes

Only some combinations of picture sizes and resampling modes are available. The M2 MPEG device does not support resampling, but the Triangle Engine can resample the decoded video frames.

See Also

`FMVOpenVideo()`, `FMVCreateIOReq()`.

FMVWriteBuffer

Write a buffer of compressed data to the MPEG video device.

Synopsis

```
Err FMVWriteBuffer(FMVDeviceHandle *device, void *buffer,
    int32 numBytes, Item ioReqItem, FMVIOReqOptionsPtr
    fmvOptionsPtr,
    uint32 fmvFlags, uint32 ptsValue, uint32 userData)
```

Description

Begin an asynchronous write of a buffer of compressed data, with optional flags and a PTS, to the MPEG video decoder device. When the I/O completes, your task will receive a signal. The signal bit you get is the one you passed to `FMVCreateIOReq()`.

See `FMVGetPTS()` for an explanation about the PTS value and `userData`.

Arguments

`device`

An open video `FMVDeviceHandle*`.

`buffer`

Address of the buffer of compressed data to write to an MPEG device.

`numBytes`

number of data bytes in the buffer.

`ioReqItem`

MPEG video IOReq Item from `FMVCreateIOReq()`.

`fmvOptionsPtr`

ptr to `FMVIOReqOptions` storage. This is used to pass PTS and flags to the MPEG decoder. Its contents on entry don't matter, but the storage must remain accessible to the MPEG device until this I/O request completes. NULL is ok if you have no PTS or flags to pass in.

`fmvFlags`

MPEG driver flags such as a combination of `FMVValidPTS`, `FMVPTSHIGHBIT`, `FMV_END_OF_STREAM_FLAG`, `FMV_DISCONTINUITY_FLAG`, and `FMV_FLUSH_FLAG`.

`ptsValue`

The low order 32 PTS bits; irrelevant if `!(fmvFlags & FMVValidPTS)`.

`userData`

A user data value to pass through the decoder along with `ptsValue`.

Return Value

status code from `SendIO()`.

Implementation

Link library call implemented in `libsubscriber.a`.

Associated Files

"fmvdriverinterface.h", libsubscriber.a

See Also

FMVOpenVideo(), FMVCreateIOReq(), FMVReadVideoBuffer(), FMVGetPTS().

DisposeDataAcq

Shuts down a data acquisition thread.

Synopsis

```
Err DisposeDataAcq(Item dataAcqMsgPort)
```

Description

Asks the data acquisition thread to clean up and shut down.

You should disconnect a data acquisition thread (via `DSConnect()`) before disposing it. After disconnecting, the data acquisition thread will continue to return the flushed buffers to the streamer as those aborted I/O operations complete.

`DisposeDataAcq()` is a synchronous operation, that is, it waits until the data acquisition thread is done communicating with the data stream thread and done shutting down.

The proper way to shut down the streamer is to disconnect its data acq (this begins decoupling the two threads), then dispose the data acq (this waits for the data acq to finish communicating with the streamer), then dispose the data streamer.

Arguments

`dataAcqMsgPort`

The data acquisition thread's request message port.

Return Value

A Portfolio Err code.

Assumes

The data acq thread has been disconnected from the streamer thread.

Implementation

Streaming library call.

Associated Files

`<:streaming:datastream.h>`, `libds.a`

See Also

`NewDataAcq()`, `DSConnect()`

DisposeDataStream

Shuts down a data stream thread.

Synopsis

```
Err DisposeDataStream(Item msgItem, DSStreamCBPtr streamCBPtr)
```

Description

Asks the streamer to shut down. The streamer will ask each subscriber to shut down and will then shut itself down. Currently, the client is responsible for shutting down the Data Acq module.

This sends a synchronous request to the Data Streamer to perform the operation. It returns an Err code. If there are no Kernel errors with the message send and reply itself, this returns the Streamer error code from the message reply.

Arguments

msgItem

Item to use for the request message to the Data Stream thread.

streamCBPtr

Pointer to the stream context block.

Return Value

kDSBadPtrErr

streamCBPtr is NULL. (The caller doesn't have to check.)

Or other Portfolio error code.

Assumes

The stream is stopped and the DataStream thread is disconnected from data acquisition threads (see `DSConnect()`) and in a clean state where it has returned messages and other resources that were passed to it.

Implementation

Streaming library call.

Associated Files

<:streaming:datastream.h>, libds.a

See Also

`NewDataStream()`, `DSConnect()`, `DisposeDataAcq()`

CreateBufferList

Allocates a stream buffer list for use by the streamer.

Synopsis

```
DSDDataBufPtr CreateBufferList(int32 numBuffers, int32 bufferSize)
```

Description

CreateBufferList allocates a data streamer input buffer list. It first allocates (via AllocMemAligned() with MEMTYPE_TRACKSIZE) one big memory block for the array of DSDDataBuf structures and the array of data buffers. It ensures that each data buffer is aligned for optimal DMA and CPU cache access. It then links up the DSDDataBuf structures (by their 'next' fields) and points each at its aligned buffer (by its 'streamData' field).

The resulting buffer list can then be passed to NewDataStream(). The streamer will use these buffers for reading blocks of data from the input device.

To later deallocate the buffer list, call FreeMemTrack(), passing the result of CreateBufferList--the first DSDDataBufPtr.

This procedure allocates the entire buffer list in one contiguous memory block. You don't have to do it this way. You could allocate each buffer separately, or use another memory manager, or whatever. You just need to how to deallocate them, in this case: FreeMemTrack(firstBp).

Notes

Unlike previous versions of the data streamer, the DSDDataBuf structures are separate from the data buffers themselves. So anytime the streamer is quiescent--that is, after a stop-flush operation completes--you can temporarily reuse the buffer memory (e.g. for transition effects buffers) as long as you don't smash the DSDDataBuf structures.

If CreateBufferList() returns non-NULL firstBp, firstBp->streamData points to the buffer memory. The buffer memory will be at least numBuffers * bufferSize bytes long.

Arguments

numBuffers

The number of buffers to allocate. This must be at least 2.

bufferSize

The data size of each buffer, in bytes. This MUST AGREE with the stream's block size. Typical buffer sizes are 32K to 96K bytes.

Return Value

A pointer to the first buffer in the list, suitable for passing to NewDataStream() and later to FreeMemTrack(). Or NULL, if numBuffers < 2. Or NULL, if CreateBufferList() couldn't allocate the memory.

Implementation

Library code in libds.a.

Associated Files

<:streaming:datastream.h>, <:streaming:dserror.h>, <:streaming:preparestream.h>

See Also

`AllocMemAligned()`, `FreeMemTrack()`, `NewDataStream()`

FillPoolWithMsgItems

Initialize a MemPool of GenericMsg entries.

Synopsis

```
bool FillPoolWithMsgItems(MemPoolPtr memPool, Item replyPort)
```

Description

Given a MemPool of GenericMsg entries, initialize each entry by creating a Message Item and storing that in the entry's msgItem field. If it fails to initialize ALL the entries, this will clean up after itself and return FALSE.

Arguments

memPool
The MemPool of GenericMsg entries to initialize.

replyPort
Reply port for each entry's Message Item.

Return Value

TRUE if successful, FALSE if not.

Implementation

Streaming library call.

Associated Files

<.:streaming:datastreamlib.h>, libds.a

See Also

<.:streaming:mempool.h>, ForEachFreePoolMember()

FindAndLoadStreamHeader

Load a stream file's stream header chunk into memory.

Synopsis

```
Err FindAndLoadStreamHeader(DSHeaderChunkPtr headerPtr,  
    char *fileName)
```

Description

Open a stream file, check that it starts with a stream header chunk, load the header into memory, and close the stream file. (The streamer's DataAcq will later reopen the file.)

Despite its name, this procedure does no searching.

Arguments

headerPtr

Points to a DSHeaderChunk struct allocated by the client. FindAndLoadStreamHeader() will store the stream's header structure (if there is one) into *headerPtr.

fileName

The name of the DS stream file to read a header from.

Return Value

An error code. In particular, the value kDSHeaderNotFound (defined in *<:streaming:derror.h>*) means FindAndLoadStreamHeader() didn't find a stream header in this file. In that case, FindAndLoadStreamHeader() will print a warning message and the application can substitute default settings.

Assumes

A stream file's stream header chunk begins at the first byte of the file.

Implementation

Library code in libds.a.

Associated Files

<:streaming:datastream.h>, *<:streaming:derror.h>*, *<:streaming:preparestream.h>*

NewDataAcq

Instantiates a new data acquisition thread.

Synopsis

```
Item NewDataAcq(char *fileName, int32 deltaPriority)
```

Description

Instantiates a new data acquisition thread. This creates the thread, allocates all its resources, and waits for initialization to complete. Call `DisposeDataAcq()` to ask the data acq thread to clean up and exit.

This temporarily allocates and frees a signal in the caller's thread. Currently, 4096 bytes are allocated for the streamer thread's stack.

Arguments

`fileName`
Name of the stream file to open.

`deltaPriority`
Task priority for the streamer thread, relative to the caller.

Return Value

The new DataAcq thread's request message port, or a Portfolio Err code such as:

`kDSNoMemErr`
Could not allocate enough memory.

`kDSSignalErr`
Problem sending/receiving a signal.

`kDSInitErr`
Other initialization error.

`kDSNoSignalErr`
Couldn't allocate a Signal to synchronize with the spawning thread.

Implementation

Streaming library call.

Associated Files

`<:streaming:datastream.h>`, `libds.a`

See Also

`DisposeDataAcq()`

NewDataStream

Instantiates a new data stream thread.

Synopsis

```
Item NewDataStream(DSStreamCBPtr *pCtx, void *bufferListPtr,  
    uint32 bufferSize, int32 deltaPriority, int32 numSubsMsgs)
```

Description

Instantiates a new data stream (server) thread. This creates the thread, allocates all its resources, and waits for initialization to complete. Call `DisposeDataStream()` to ask the streamer thread to clean up and exit.

This currently returns the streamer's context-block (instance variables) pointer via `*pCtx`. The caller should not peek or poke through this pointer. But currently, there are many procedures which require a `DSStreamCBPtr` argument.

This temporarily allocates and frees a signal in the caller's thread. Currently, 4096 bytes are allocated for the streamer thread's stack.

Arguments

`pCtx`

Address to store the new stream context pointer.

`bufferListPtr`

Pointer to a list of input buffers for the streamer to use.

Currently, the streamer uses the first part of each of these structures as a header to track the buffer, and the rest as an I/O buffer. In the future, we'll separate these two, so the buffers themselves can be reused while streaming is quiescent.

`bufferSize`

Size of each buffer in the list. This **MUST** be a whole number of disc blocks and it **MUST** agree with the woven stream block size.

`deltaPriority`

Task priority for the streamer thread, relative to the caller.

`numSubsMsgs`

Number of subscriber messages that the streamer should allocate.

Return Value

The Data Stream thread's request message port, or a negative Portfolio error code such as:

`kDSNoMemErr`

Could not allocate enough memory.

`kDSBadBufAlignErr`

One or more of the buffers passed in is not quadbyte aligned.

`kDSRangeErr`

Not enough buffers passed in, or other input-parameter range error.

`kDSSignalErr`

Problem sending/receiving a signal.

kDSInitErr

Other initialization error.

kDSNoSignalErr

Couldn't allocate a Signal to synchronize with the spawning thread.

kDSNoMsgErr

Couldn't allocate a needed Message Item.

This procedure also returns a pointer to the new streamer context in *pCtx.

Implementation

Streaming library call.

Associated Files

<:streaming:datastream.h>, libds.a

See Also

DisposeDataStream()

NewDataSubscriber

Instantiates a DATASubscriber.

Synopsis

```
Err NewDataSubscriber(DataContextPtr *ctxPtrPtr,  
    DSStreamCBPtr streamCBPtr, int32 deltaPriority, Item msgItem)
```

Description

Instantiates a new DATASubscriber. This creates the subscriber thread, waits for initialization to complete, and signs it up for a subscription with the Data Stream parser thread. All channels are enabled.

Arguments

ctxPtrPtr

A pointer to a variable where the data subscriber context handle will be stored. If initialization fails, the variable is set to NULL.

streamCBPtr

The new subscriber will subscribe to this Streamer.

deltaPriority

The priority (relative to the caller) for the subscriber thread.

msgItem

A Message Item to use temporarily for a synchronous DSSubscribe() call.

Return Value

Returns zero for success, or a negative error code for failure.

Implementation

Streaming subscriber library call.

Associated Files

<:streaming:datasubscriber.h>, libsubscriber.a

NewEZFlixSubscriber

Instantiates an EZFlixSubscriber.

Synopsis

```
Item NewEZFlixSubscriber(DSStreamCBPtr streamCBPtr,  
    int32 deltaPriority, Item msgItem, uint32 numChannels,  
    EZFlixContextPtr *pContextPtr)
```

Description

Instantiates a new EZFlixSubscriber. This creates the subscriber thread, waits for initialization to complete, *and* signs it up for a subscription with the Data Stream parser thread. This returns the new subscriber's request message port Item number or a negative error code. The subscriber will clean itself up and exit when it receives a kStreamOpClosing or kStreamOpAbort message. (If DSSubscribe() fails, the streamer will send a kStreamOpAbort message to the subscriber-wannabe.)

This temporarily allocates and frees a signal in the caller's thread. Currently, 4096 bytes are allocated for the subscriber's stack.

Arguments

streamCBPtr

The new subscriber will subscribe to this Streamer.

deltaPriority

The priority (relative to the caller) for the subscriber thread.

msgItem

A Message Item to use temporarily for a synchronous DSSubscribe() call.

numChannels

The number of channels to support. This puts an upper bound on the channel numbers that the EZFlix Subscriber will listen to. There's also an implementation max, currently set to 8.

Return Value

A positive Item number of the new subscriber's request message port or a negative error code indicating initialization errors, e.g. kDSNoMemErr (couldn't allocate memory), kDSNoSignalErr (couldn't allocate a signal), kDSSignalErr (problem sending or receiving a signal), kDSRangeErr (parameter such as bitDepth out of range), or kDSInitErr (initialization problem, e.g. allocating IOReq Items).

Implementation

Streaming library call.

Associated Files

<.:streaming:ezflixsubscriber.h>, libsubscriber.a

NewMPEGAudioSubscriber

Create a new SAudio subscriber thread.

Synopsis

```
Item NewMPEGAudioSubscriber(DSStreamCBPtr streamCBPtr,  
                             int32 deltaPriority, Item msgItem, uint32 numberOfBuffers)
```

Description

Instantiate a new SAudioSubscriber. This creates the subscriber thread, waits for initialization to complete, *and* signs it up for a subscription with the Data Stream parser thread. This returns the new subscriber's request message port Item number or a negative error code. The subscriber will clean itself up and exit when it receives a kStreamOpClosing or kStreamOpAbort message. (If DSSubscribe() fails, the streamer will send a kStreamOpAbort message to the subscriber-wannabe.)

This temporarily allocates and frees a signal in the caller's thread. Currently, 4096 bytes are allocated for the subscriber's stack.

Arguments

streamCBPtr

Pointer to control block for data stream. The new subscriber will subscribe to this Streamer.

deltaPriority

Execution priority of new SAudio subscriber thread relative to the caller.

msgItem

A Message Item to use temporarily for a synchronous DSSubscribe() call.

numberOfBuffers

Number of audio buffers to allocate for storing decompressed data.

Return Value

A positive Item number of the new subscriber's request message port or a negative error code indicating initialization errors, e.g. kDSNoMemErr (couldn't allocate memory), kDSNoSignalErr (couldn't allocate a signal), kDSSignalErr (problem sending or receiving a signal), kDSRangeErr (parameter such as bitDepth out of range), or kDSInitErr (initialization problem, e.g. allocating IOReq Items).

Implementation

Streaming library call.

Associated Files

<:streaming:sudiosubscriber.h>, libsubscriber.a

NewMPEGVideoSubscriber

Instantiates an MPEGVideoSubscriber.

Synopsis

```
Item NewMPEGVideoSubscriber(DSStreamCBPtr streamCBPtr,  
                             int32 deltaPriority, Item msgItem, uint32 bitDepth)
```

Description

Instantiates a new MPEGVideoSubscriber. This creates the subscriber thread, waits for initialization to complete, *and* signs it up for a subscription with the Data Stream parser thread. This returns the new subscriber's request message port Item number or a negative error code. The subscriber will clean itself up and exit when it receives a kStreamOpClosing or kStreamOpAbort message. (If DSSubscribe() fails, the streamer will send a kStreamOpAbort message to the subscriber-wannabe.)

This temporarily allocates and frees a signal in the caller's thread. Currently, 4096 bytes are allocated for the subscriber's stack.

Arguments

streamCBPtr

The new subscriber will subscribe to this Streamer.

deltaPriority

The priority (relative to the caller) for the subscriber thread.

msgItem

A Message Item to use temporarily for a synchronous DSSubscribe() call.

bitDepth

The bit-depth for decoded pixels, either 16 or 24.

Return Value

A positive Item number of the new subscriber's request message port or a negative error code indicating initialization errors, e.g. kDSNoMemErr (couldn't allocate memory), kDSNoSignalErr (couldn't allocate a signal), kDSSignalErr (problem sending or receiving a signal), kDSRangeErr (parameter such as bitDepth out of range), or kDSInitErr (initialization problem, e.g. allocating IOReq Items).

Implementation

Streaming library call.

Associated Files

<:streaming:mpegvideosubscriber.h>, libsubscriber.a

NewSAudioSubscriber

Create a new SAudio subscriber thread.

Synopsis

```
Item NewSAudioSubscriber(DSStreamCBPtr streamCBPtr,  
                          int32 deltaPriority, Item msgItem)
```

Description

Instantiate a new SAudioSubscriber. This creates the subscriber thread, waits for initialization to complete, *and* signs it up for a subscription with the Data Stream parser thread. This returns the new subscriber's request message port Item number or a negative error code. The subscriber will clean itself up and exit when it receives a kStreamOpClosing or kStreamOpAbort message. (If DSSubscribe() fails, the streamer will send a kStreamOpAbort message to the subscriber-wannabe.)

This temporarily allocates and frees a signal in the caller's thread. Currently, 4096 bytes are allocated for the subscriber's stack.

Arguments

streamCBPtr

Pointer to control block for data stream. The new subscriber will subscribe to this Streamer.

deltaPriority

Execution priority of new SAudio subscriber thread relative to the caller.

msgItem

A Message Item to use temporarily for a synchronous DSSubscribe() call.

Return Value

A positive Item number of the new subscriber's request message port or a negative error code indicating initialization errors, e.g. kDSNoMemErr (couldn't allocate memory), kDSNoSignalErr (couldn't allocate a signal), kDSSignalErr (problem sending or receiving a signal), kDSRangeErr (parameter such as bitDepth out of range), or kDSInitErr (initialization problem, e.g. allocating IOReq Items).

Implementation

Streaming library call.

Associated Files

<:streaming:sudiosubscriber.h>, libsubscriber.a

DSConnect

Connects/replaces/disconnects a data acquisition client to the stream.

Synopsis

```
Err DSConnect(Item msgItem, DSRequestMsgPtr reqMsg,  
              DSStreamCBPtr streamCBPtr, Item acquirePort)
```

Description

Asks the streamer to connect/replace/disconnect a data acquisition client. If the streamer is currently connected, it will send a disconnect message to the current data acquisition client. Then, (if `acquirePort != 0`) it will switch to the new data acquisition client and send it a connect message.

On connect, the streamer automatically sets the presentation clock. On disconnect, the streamer flushes all data queued at the old data acq and at subscribers.

The streamer replies to the connect request (which allows a "synchronous" `DSConnect()` call to return), when the changeover of data acqs is effective, but it doesn't wait for the old data acq to finish returning flushed buffers. The disconnected data acq will return the flushed buffers to the streamer as their aborted I/O operations complete.

The proper way to shut down the streamer is to disconnect its data acq (this begins decoupling the two threads), then dispose the data acq (this waits for the data acq to finish communicating with the streamer), then dispose the data streamer.

`DSConnect()` sends a synchronous request (if `reqMsg == NULL`) or an asynchronous request (if `reqMsg != NULL`) to the Data Streamer to perform the operation. ("Synchronous" means the procedure waits for the Streamer's reply.) It returns any error code arising from the message send. For a synchronous request with no messaging errors, this returns the Streamer's error code (from the message's `msg_Result` field).

Arguments

`msgItem`

Item to use for the request message to the Data Stream thread.

`reqMsg`

Pointer to the `DSRequestMsg` message struct to fill in and send as an asynchronous request to the Data Streamer. This struct must remain allocated and available to the data stream thread until it replies to the message. A value of `NULL` means use a stack-allocated `DSRequestMsg` and do a synchronous operation.

`streamCBPtr`

Pointer to the stream context block.

`acquirePort`

Message port of the DataAcq to connect to, or 0 to disconnect.

Return Value

`kDSBadPtrErr`

`streamCBPtr` is `NULL`. (The caller doesn't have to check.)

Or other Portfolio error code.

Implementation

Streaming library call.

Associated Files

<:streaming:datastreamlib.h>, libds.a

See Also

<:streaming:datastream.h>, NewDataAcq(), DisposeDataStream(), DisposeDataAcq()

DSControl

Sends a control message to a data stream subscriber.

Synopsis

```
Err DSControl(Item msgItem, DSRequestMsgPtr reqMsg,  
              DSStreamCBPtr streamCBPtr, DSDataType streamType,  
              int32 userDefinedOpcode, void *userDefinedArgPtr)
```

Description

Sends a control message to a data stream subscriber.

`DSControl()` sends a synchronous request (if `reqMsg == NULL`) or an asynchronous request (if `reqMsg != NULL`) to the Data Streamer to perform the operation. ("Synchronous" meaning the procedure waits for the Streamer's reply.) It returns any error code arising from the message send. For a synchronous request with no messaging errors, this returns the Streamer's error code (from the message's `msg_Result` field).

Arguments

`msgItem`

Item to use for the request message to the Data Stream thread.

`reqMsg`

Pointer to the `DSRequestMsg` message struct to fill in and send as an asynchronous request to the Data Streamer. This struct must remain allocated and available to the data stream thread until it replies to the message. A value of `NULL` means use a stack-allocated `DSRequestMsg` and do a synchronous operation.

`streamCBPtr`

Pointer to the stream context block.

`streamType`

Elementary stream data type (subscriber type) to control.

`userDefinedOpcode`

Defined by the subscriber.

`userDefinedArgPtr`

Defined by the subscriber; depends on `userDefinedOpcode`.

Return Value

`kDSSubNotFoundErr`

The stream has no subscriber for this data type.

`kDSBadPtrErr`

`streamCBPtr` is `NULL`. (The caller doesn't have to check.)

Or other Portfolio error code.

Implementation

Streaming library call.

Associated Files

`<:streaming:datastreamlib.h>`, `libds.a`

See Also

`<:streaming:datastream.h>`

DSGetChannel

Read the status of a subscriber's logical channel.

Synopsis

```
Err DSGetChannel(Item msgItem, DSRequestMsgPtr reqMsg, DSStreamCBPtr  
    streamCBPtr, DSDataType streamType, uint32 channelNumber,  
    uint32 *channelStatusPtr)
```

Description

Read the status of a subscriber's logical channel into *channelStatusPtr.

DSGetChannel() sends a synchronous request (if reqMsg == NULL) or an asynchronous request (if reqMsg != NULL) to the Data Streamer to perform the operation. ("Synchronous" meaning the procedure waits for the Streamer's reply.) It returns any error code arising from the message send. For a synchronous request with no messaging errors, this returns the Streamer's error code (from the message's msg_Result field).

Arguments

msgItem

Item to use for the request message to the Data Stream thread.

reqMsg

Pointer to the DSRequestMsg message struct to fill in and send as an asynchronous request to the Data Streamer. This struct must remain allocated and available to the data stream thread until it replies to the message. A value of NULL means use a stack-allocated DSRequestMsg and do a synchronous operation.

streamCBPtr

Pointer to the stream context block.

streamType

Elementary stream data type (subscriber type) to get status for.

channelNumber

Number of the logical channel to read status for.

channelStatusPtr

Address to store the channel status bits.

Return Value

kDSSubNotFoundErr

The stream has no subscriber for this data type.

kDSBadPtrErr

streamCBPtr is NULL. (The caller doesn't have to check.)

Or other Portfolio error code.

Implementation

Streaming library call.

Associated Files

<:streaming:datastreamlib.h>, libds.a

See Also

`<:streaming:datastream.h>`, `DSSetChannel()`.

DSGoMarker

Asks the streamer to branch within the stream.

Synopsis

```
Err DSGoMarker(Item msgItem, DSRequestMsgPtr reqMsg,  
                DSStreamCBPtr streamCBPtr, uint32 markerValue, uint32 options)
```

Description

Asks the streamer to branch within the stream. Stream positions are normally defined by a Marker Table embedded in the stream, but you can also branch to an absolute position in the stream.

The options argument specifies the branch type, i.e. how to use the markerValue argument to find the branch destination. All branch types except GOMARKER_ABSOLUTE require the DataAcq to have received a Marker Table chunk from the stream.

When you use the Weaver, you can tell it where in the stream you want marker points, in units of Audio ticks of stream time. The Weaver will generate a marker table sorted by stream time. Each marker table entry gives the stream byte position and stream presentation time of one marker point.

With branching, there are issues to consider such as flushing part or all of the data flow pipeline and covering CD seek times.

DSGoMarker() normally performs a "flush branch", which means it flushes data that's queued at the subscribers in order to reach the destination ASAP. But you can bitor GOMARKER_NO_FLUSH_FLAG into the options argument to ask the streamer to instead perform a "butt-joint branch" where the subscribers play out their queued data before playing the post-branch data. A flush branch produces an immediate reaction but does no seek covering. A non-flush branch has a delayed response time but does some seek covering.

WARNING: DSGoMarker(GOMARKER_NO_FLUSH_FLAG) is not well defined since the request is asynchronous to the stream but it asks the streamer to finish playing whatever data has been delivered to subscribers. Since the streamer doesn't know when the old data will finish, it can't adjust the presentation clock over this discontinuity. If an audio subscriber is playing data on its "clock channel", it'll set the clock. Otherwise the stream will appear frozen while all subscribers wait for the clock to get past the discontinuity.

The Data Stream thread will react to a GOTO chunk in the stream much like it reacts to a DSGoMarker() call. However a GOTO chunk is at a well-defined place in the stream and it has its own presentation time stamp, so the streamer could adjust the clock after the branch. (NOTE: The streamer does not yet adjust the clock after a non-flush branch.) A GOTO chunk normally does a non-flush branch in order, i.e. plays out the data that preceeds it in the stream.

DSGoMarker() sends a synchronous request (if reqMsg == NULL) or an asynchronous request (if reqMsg != NULL) to the Data Streamer to perform the operation. ("Synchronous" means the procedure waits for the Streamer's reply.) It returns any error code arising from the message send. For a synchronous request with no messaging errors, this returns the Streamer's error code (from the message's msg_Result field).

Arguments

msgItem

Item to use for the request message to the Data Stream thread.

reqMsg

Pointer to the DSRequestMsg message struct to fill in and send as an asynchronous request

to the DataStreamer. This struct must remain allocated and available to the data stream thread until it replies to the message. A value of NULL means use a stack-allocated DSRequestMsg and do a synchronous operation.

streamCBPtr

Pointer to the Stream context block.

markerValue

This argument supplies a branch amount. Depending on the options argument, markerValue could be an absolute or relative count of presentation clock ticks, marker numbers, or file bytes.

options

This argument specifies the branch type (how to interpret the markerValue argument) and whether to take a flush branch or a butt-joint branch. See "Options", below.

Options

options choice	use of markerValue
-----	-----
GOMARKER_ABSOLUTE markerValue position block.)	branch to the absolute file byte position (No marker table needed.) (The destination no longer needs to be the start of a stream block.)
GOMARKER_FORWARD markers	find the first marker at or after the current presentation time, count forward markerValue in the marker table, and branch to that marker
GOMARKER_BACKWARD markers	find the first marker at or after the current presentation time, count backward markerValue in the marker table, and branch to that marker
GOMARKER_ABS_TIME markerValue, in	branch to the first marker at or after audio ticks
GOMARKER_FORW_TIME current ticks	branch to the first marker at or after the presentation time plus markerValue, in audio ticks
GOMARKER_BACK_TIME current ticks	branch to the first marker at or after the presentation time minus markerValue, in audio ticks
GOMARKER_NAMED	char* name of destination marker [UNIMPLEMENTED]
GOMARKER_NUMBER into	branch to marker number markerValue, an index into

the
marker table ranging from 0 to the number of
markers - 1

options flag	meaning
--------------	---------

GOMARKER_NO_FLUSH_FLAG	Don't flush data queued at the subscribers.
------------------------	--

[See the WARNING under Description,
above.]

Return Value

kDSBadPtrErr

streamCBPtr is NULL. (The caller doesn't have to check.)

kDSUnImplemented

Unimplemented branch type (GOMARKER_NAMED).

kDSEndOfFileErr

Tried to branch beyond the end of the file via GOMARKER_ABSOLUTE or a bum
marker.

kDSBranchNotDefined

Undefined branch destination, in particular, can't branch to a marker if the DataAcq
hasn't received a marker table from the stream.

kDSRangeErr

Parameter out of range: undefined options choice or GOMARKER_NUMBER,
GOMARKER_FORWARD, or GOMARKER_BACKWARD tried to index beyond the
marker table's bounds. (The first marker is marker number 0.)

Or other Portfolio error code.

Implementation

Streaming library call.

Associated Files

<:streaming:datastreamlib.h>, libds.a

See Also

<:streaming:datastream.h>

DSIsRunning

Returns TRUE if the Data Stream's presentation clock is currently running.

Synopsis

```
bool DSIsRunning(DSStreamCBPtr streamCBPtr)
```

Description

Returns TRUE if the Data Stream's presentation clock is currently running, i.e. if the Data Stream thread is currently delivering data to subscribers.

Arguments

streamCBPtr
Pointer to the stream context block.

Return Value

TRUE if the Data Stream's presentation clock is currently running. FALSE if not.

Implementation

Streaming library call.

Associated Files

<:streaming:datastreamlib.h>, libds.a

See Also

DSStartStream(), DSStopStream(), DSGetPresentationClock().

DSPreRollStream

Start prefilling empty buffers with data.

Synopsis

```
Err DSPreRollStream(Item msgItem, DSRequestMsgPtr reqMsg,  
    DSStreamCBPtr streamCBPtr, uint32 asyncBufferCnt)
```

Description

Asks the Data Streamer to start filling its buffers from the input device. You call this before starting stream playback.

The Data Streamer will wait for all but `asyncBufferCnt` of the buffers to return from `DataAcq` before replying to this request msg. So preroll is more than a hint. This is esp. important with MPEG.

We give the client control over the desired preroll buffer level because waiting for ALL the buffers to come back before starting playback could easily miss a disc rev. Tune it empirically. Start with something like 2.

`DSPreRollStream()` sends a synchronous request (if `reqMsg == NULL`) or an asynchronous request (if `reqMsg != NULL`) to the Data Streamer to perform the operation. ("Synchronous" meaning the procedure waits for the Streamer's reply.) It returns any error code arising from the message send. For a synchronous request with no messaging errors, this returns the Streamer's error code (from the message's `msg_Result` field).

Arguments

`msgItem`

Item to use for the request message to the Data Stream thread.

`reqMsg`

Pointer to the `DSRequestMsg` message struct to fill in and send as an asynchronous request to the DataStream. This struct must remain allocated and available to the data stream thread until it replies to the message. A value of `NULL` means use a stack-allocated `DSRequestMsg` and do a synchronous operation.

`streamCBPtr`

Pointer to the stream context block.

`asyncBufferCnt`

This many buffers may be filled asynchronously, that is, after the Streamer replies to the request message. If you start playback after all but a couple buffers are filled, there'll be a smaller chance of missing a disc rev.

Return Value

`kDSBadPtrErr`

`streamCBPtr` is `NULL`. (The caller doesn't have to check.)

Or other Portfolio error code.

Caveats

This procedure expects to be called when the stream is not "running" and not at EOF, but it doesn't check.

Implementation

Streaming library call.

Associated Files

<:streaming:datastreamlib.h>, libds.a

See Also

<:streaming:datastream.h>, DSStartStream()

DSSetChannel

Sets the status of a subscriber's logical stream channel.

Synopsis

```
Err DSSetChannel(Item msgItem, DSRequestMsgPtr reqMsg,  
    DSStreamCBPtr streamCBPtr, DSDataType streamType,  
    uint32 channelNumber, uint32 channelStatus, uint32 mask)
```

Description

Sets the status of a subscriber's logical stream channel.

`DSSetChannel()` sends a synchronous request (if `reqMsg == NULL`) or an asynchronous request (if `reqMsg != NULL`) to the Data Streamer to perform the operation. ("Synchronous" meaning the procedure waits for the Streamer's reply.) It returns any error code arising from the message send. For a synchronous request with no messaging errors, this returns the Streamer's error code (from the message's `msg_Result` field).

Arguments

`msgItem`

Item to use for the request message to the Data Stream thread.

`reqMsg`

Pointer to the `DSRequestMsg` message struct to fill in and send as an asynchronous request to the Data Streamer. This struct must remain allocated and available to the data stream thread until it replies to the message. A value of `NULL` means use a stack-allocated `DSRequestMsg` and do a synchronous operation.

`streamCBPtr`

Pointer to the stream context block.

`streamType`

Elementary stream data type (subscriber type) to set status for.

`channelNumber`

Logical channel number to set status for.

`channelStatus`

New status bits for the channel.

`mask`

Determines which status bits to set.

Return Value

`kDSSubNotFoundErr`

The stream has no subscriber for this data type.

`kDSBadPtrErr`

`streamCBPtr` is `NULL`. (The caller doesn't have to check.)

Or other Portfolio error code.

Implementation

Streaming library call.

Associated Files

<:streaming:datastreamlib.h>, libds.a

See Also

<:streaming:datastream.h>, DSGetChannel () .

DSStartStream

Starts a stream playing.

Synopsis

```
Err DSStartStream(Item msgItem, DSRequestMsgPtr reqMsg,  
                  DSStreamCBPtr streamCBPtr, uint32 startOptions)
```

Description

Asks the DataStreamer to start playback and start the presentation clock.

You can pause the stream by calling `DSStopStream(..., SOFT_NOFLUSH)` and resume it by calling `DSStartStream(..., SOFT_NOFLUSH)`. In that case, the streamer will pause and resume the presentation clock. If you start the stream after flushing it, the streamer will automatically set the presentation clock.

`DSStartStream()` sends a synchronous request (if `reqMsg == NULL`) or an asynchronous request (if `reqMsg != NULL`) to the Data Streamer to perform the operation. ("Synchronous" meaning the procedure waits for the Streamer's reply.) It returns any error code arising from the message send. For a synchronous request with no messaging errors, this returns the Streamer's error code (from the message's `msg_Result` field).

Arguments

`msgItem`

Item to use for the request message to the Data Stream thread.

`reqMsg`

Pointer to the `DSRequestMsg` message struct to fill in and send as an asynchronous request to the DataStreamer. This struct must remain allocated and available to the data stream thread until it replies to the message. A value of `NULL` means use a stack-allocated `DSRequestMsg` and do a synchronous operation.

`streamCBPtr`

Pointer to the stream context block.

`startOptions`

`SOPT_FLUSH` to flush the stream buffers before starting, otherwise `SOFT_NOFLUSH`.

Return Value

`kDSWasRunningErr`

The stream was already playing.

`kDSBadPtrErr`

`streamCBPtr` is `NULL`. (The caller doesn't have to check.)

Or other Portfolio error code.

Implementation

Streaming library call.

Associated Files

<:streaming:datastreamlib.h>, libds.a

See Also

`<:streaming:datastream.h>, DSPreRollStream(), DSStopStream()`

DSStopStream

Stops stream playback.

Synopsis

```
Err DSStopStream(Item msgItem, DSRequestMsgPtr reqMsg, DSStreamCBPtr  
    streamCBPtr, uint32 stopOptions)
```

Description

Asks the DataStreamer to stop playback and stop the presentation clock.

You can pause the stream by calling `DSStopStream(..., SOFT_NOFLUSH)` and resume it by calling `DSSstartStream(..., SOFT_NOFLUSH)`. In that case, the streamer will pause and resume the presentation clock. If you asked to flush the stream via these calls, the streamer will automatically reset the presentation clock when it starts stream playback.

After pausing, you might want to call `DSPreRollStream()` to refill all stream buffers. But it might not matter much because few buffers will return from subscribers while the stream is paused.

If you stop the stream with the `SOPT_FLUSH` option, the streamer will relinquish its use of the stream data buffers (but not the `DSDataBuf` structures that point to the data buffers) so you can reuse the memory.

`DSStopStream()` sends a synchronous request (if `reqMsg == NULL`) or an asynchronous request (if `reqMsg != NULL`) to the Data Streamer to perform the operation. ("Synchronous" meaning the procedure waits for the Streamer's reply.) It returns any error code arising from the message send. For a synchronous request with no messaging errors, this returns the Streamer's error code (from the message's `msg_Result` field).

Arguments

`msgItem`

Item to use for the request message to the Data Stream thread.

`reqMsg`

Pointer to the `DSRequestMsg` message struct to fill in and send as an asynchronous request to the DataStreamer. This struct must remain allocated and available to the data stream thread until it replies to the message. A value of `NULL` means use a stack-allocated `DSRequestMsg` and do a synchronous operation.

`streamCBPtr`

Pointer to the stream context block.

`stopOptions`

`SOPT_FLUSH` to flush the stream buffers before starting, otherwise `SOFT_NOFLUSH`.

Return Value

`kDSBadPtrErr`

`streamCBPtr` is `NULL`. (The caller doesn't have to check.)

Or other Portfolio error code.

Implementation

Streaming library call.

Associated Files

<:streaming:datastreamlib.h>, libds.a

See Also

<:streaming:datastream.h>, DSStartStream()

DSSubscribe

Add/replace/remove a data stream subscriber.

Synopsis

```
Err DSSubscribe(Item msgItem, DSRequestMsgPtr reqMsg,  
                DSStreamCBPtr streamCBPtr, DSDataType dataType,  
                Item subscriberPort)
```

Description

Sends a request to the Data Stream thread to add/replace/remove a subscriber. The Data Streamer will start sending data chunks of type `dataType` to `subscriberPort` (or discarding them, if `subscriberPort == 0`).

If there's currently a subscriber for `dataType`, the Data Streamer will unsubscribe it and send it a `kStreamOpClosing` message so it'll know to close down. If the new `subscriberPort` is not 0, the Data Streamer will subscribe it and send it a `kStreamOpOpening` message, unless it can't add the subscriber, in which case it'll send it a `kStreamOpAbort` message (to close down) and return an error code.

`DSSubscribe()` sends a synchronous request (if `reqMsg == NULL`) or an asynchronous request (if `reqMsg != NULL`) to the Data Streamer to perform the operation. ("Synchronous" meaning the procedure waits for the Streamer's reply.) It returns any error code arising from the message send. For a synchronous request with no messaging errors, this returns the Streamer's error code (from the message's `msg_Result` field).

Arguments

`msgItem`

Item to use for the request message to the Data Stream thread.

`reqMsg`

Pointer to the `DSRequestMsg` message struct to fill in and send as an asynchronous request to the Data Streamer. This struct must remain allocated and available to the data stream thread until it replies to the message. A value of `NULL` means use a stack-allocated `DSRequestMsg` and do a synchronous operation.

`streamCBPtr`

Pointer to the stream context block.

`dataType`

Elementary stream data type to (un)subscribe to.

`subscriberPort`

Subscriber's Message port to receive data and control messages.

Return Value

`kDSSubMaxErr`

This is returned by the streamer (in this call if a synchronous request; in the message if an asynchronous request) if the streamer has no room in its subscriber table for another subscriber.

`kDSBadPtrErr`

`streamCBPtr` is `NULL`. (The caller doesn't have to check.)

Or other Portfolio error code.

Implementation

Streaming library call.

Associated Files

<:streaming:datastreamlib.h>, libds.a

See Also

<:streaming:datastream.h>

DSWaitEndOfStream

Register for end-of-stream notification.

Synopsis

```
Err DSWaitEndOfStream(Item msgItem, DSRequestMsgPtr reqMsg,  
    DSStreamCBPtr streamCBPtr)
```

Description

This asks the Data Streamer for end-of-stream notification. It works by sending a message; the Streamer will reply to this message when it reaches EOS. When you get the message reply, check the `msg_Result` field. Afterwards, call `DSWaitEndOfStream()` again if you want to re-register for end-of-stream notification.

A Data Streamer only supports one EOS registrant at a time. So `DSWaitEndOfStream()` will replace the previous registrant, replying to the previous registrant with `kDSEOSRegistrationErr`.

This sends a synchronous request (if `reqMsg == NULL`) or an asynchronous request (if `reqMsg != NULL`) to the Data Streamer to perform the operation. ("Synchronous" means the procedure waits for the Streamer's reply--which in this case means it really will wait for the End Of Stream!) Unlike all other streamer request messages, a pending end-of-stream notification request message doesn't lock out other streamer request messages.

This returns any error code arising from the message send. For a synchronous request with no messaging errors, this returns the Streamer's error code (from the message's `msg_Result` field).

Arguments

`msgItem`

Item to use for the request message to the Data Stream thread.

`reqMsg`

Pointer to the `DSRequestMsg` message struct to fill in and send as an asynchronous request to the Data Streamer. This struct must remain allocated and available to the data stream thread until it replies to the message. A value of `NULL` means use a stack-allocated `DSRequestMsg` and do a synchronous operation.

`streamCBPtr`

Pointer to the stream context block

Return Value

`kDSBadPtrErr`

`streamCBPtr` is `NULL`. (The caller doesn't have to check.)

Or other Portfolio error code.

EOS message reply's `msg_Result`

`kDSNoErr == 0`

Normal end of stream playback. This means the streamer has delivered the last of the stream's data to subscribers and the subscribers have processed it enough to return the chunks to the streamer. The subscribers might still be playing out the last decoded data. The stream clock is still running. The client might next call `DSStopStream()` to finish stopping or `DSGoMarker()` to continue playback at a different point.

`kDSSTOPChunk`

The stream stopped at a STOP chunk. You might want to call `DSPreRollStream()`. You can resume playback by calling `DSStartStream(..., SOPT_NOFLUSH)`, presumably after re-registering for EOS notification.

kDSEOSRegistrationErr

This EOS message registrant was just replaced by a new registrant.

kDSAbortErr

Stream playback was aborted due to an unrecoverable problem.

other Portfolio error code

Most likely an I/O error code from reading the stream file.

Caveats

This procedure should have been called `DSRegisterEndOfStream()`.

If the stream is already at EOS, the message will wait for the next EOS.

Implementation

Streaming library call.

Associated Files

`<:streaming:datastreamlib.h>`, `libds.a`

See Also

`<:streaming:datastream.h>`, `DSStartStream()`, `DSStopStream()`

AddTrace

Add a trace entry to a trace buffer.

Synopsis

```
void    AddTrace(TraceBufferPtr traceBuffer, int32 event,
                int32 chan, int32 value, void *ptr)
```

Background

The streamer has a simple facility for log real-time events. `AddTrace()` is called by streamer threads (historically just the subscribers) to trace high-level events with very little performance impact. `DumpRawTraceBuffer()` is called to dump the trace log when the test run is done. `DumpTraceCompletionStats()` can be called to dump <start, completion> event pairs and their durations.

This functionality is invaluable when analyzing a subscriber's performance, playback smoothness, and thread interactions.

You can use the MPW tool canon to map event ID numbers to mnemonic event names in the dumped-out log file. You can filter the log file using the MPW tool "Search" or the Unix tool "grep". Someday there'll be a high level tool for viewing and filtering log files.

Description

`AddTrace()` adds a trace entry of the given trace event ID to the given trace buffer. The other args are just values to add to the log entry, but there are restrictions on the "value" arg if you want to use the `DumpEventCompletionStats` feature. `AddTrace()` will timestamp the new log entry.

Use the debugger to set the `traceWrap` global variable to control whether `AddTrace()` treats the trace log as a wrap-around buffer or a fill-once buffer. A fill-once buffer is more useful when debugging start-up activities.

Arguments

`traceBuffer`

A trace buffer to add to. In this unfortunate design, each thread should have its own trace buffer. This will get fixed in the future, most likely by logging to LumberJack.

`event`

Trace ID event code defined as a constant in a .h file. The file `SubscriberTrace.dict` holds a canon file to map these event ID numbers back to their mnemonics.

`channel`

A value to log. Old comment: "Logical channel, -1 if indeterminate." `DumpRawTraceBuffer()` dumps this field in decimal.

`value`

Another value to log, typically an `ioReqItem` or a signal mask.

NOTE: For <start, completion> events to be used with `DumpEventCompletionStats`, the value field must be equal in the start and completion events. E.g. it could hold the `IOReqItem` in a <SendIO, I/O-done> event pair.

`ptr`

Another value to log, typically a buffer pointer or subscriber message pointer.

Caveats

This facility currently uses a separate trace buffer for each thread, and it time-stamps the entries using the audio clock. This means that tracing multiple threads will have no synchronization impact, but the events are not time-stamped at high enough resolution to re-interleave them into a single log.

In the future this facility may be rebuilt atop LumberJack. Or at least it should time-stamp events at a higher resolution (this is expensive on Opera but not on M2) or merge them into a single log using semaphores (lighter weight on M2 than on Opera) or the PowerPC reservation register (very light weight).

Implementation

Streaming library call.

Associated Files

<:streaming:subscribertraceutils.h>, libsubscriber.a

See Also

DumpRawTraceBuffer(), DumpTraceCompletionStats()

DumpRawTraceBuffer

Dump a tabular listing of all entries in a trace buffer.

Synopsis

```
Err    DumpRawTraceBuffer(TraceBufferPtr traceBuffer,
                          const char *filename)
```

Description

Dump column headings and a columnal listing of all entries in a trace buffer. (This revision dumps the entries in time order!)

You can use the MPW tool canon to map event ID numbers to mnemonic event names in the dumped-out log file. You can filter the log file using the MPW tool "Search" or the Unix tool "grep". Someday there'll be a high level tool for viewing and filtering log files.

Arguments

traceBuffer

A trace buffer with entries added by AddTrace().

filename

A filename to dump the log to.

NOTE: If subscribertraceutils.c is compiled to dump the log via printf to the debugger Terminal, the filename arg is ignored.

Caveats

By setting the USE_FOPEN and USE_RAW_FILE compile-time switches in subscribertraceutils.c, you can control whether the log gets dumped via fopen(), RawFile, or printf(). Currently, fopen() and RawFile won't write to a debugger-side file, so DumpRawTraceBuffer() is set to use printf().

In Opera, the fopen() facility is undocumented and unsupported. NOTE that fopen() and fclose() won't truncate the contents of the file. So on Opera, be sure to remote the previous log files from "/remote" before doing another test run.

Implementation

Streaming library call.

Associated Files

<.:streaming:subscribertraceutils.h>, libsubscriber.a

See Also

AddTrace(), DumpTraceCompletionStats()

DumpTraceCompletionStats

Dump stats on <start, completion> event pairs from a trace buffer.

Synopsis

```
Err    DumpTraceCompletionStats(TraceBufferPtr traceBuffer,
                                int32 startEventCode, int32 completionEventCode,
                                const char *filename)
```

Description

Find <start, completion> event pairs from a trace buffer, and dump stats on these pairs including the time duration from start to completion.

For two events to match as a <start, completion> event pair, they must have the respective event IDs (as passed to `DumpTraceCompletionStats()`), equal `aValue` fields, and the completion event must occur after the start event.

E.g. you can use an event pair for <SendIO, I/O-done> or <audio-buffer-started, audio-buffer-completed>.

Arguments

`traceBuffer`

A trace buffer with entries added by `AddTrace()`. In this obsolete scheme, each thread should have its own trace buffer. This will get fixed in the future, most likely by logging to LumberJack.

`startEventCode`

The start event's event code. `DumpTraceCompletionStats()` will find all occurrences of this start event in the trace buffer.

`completionEventCode`

The matching completion event's event code. For each start event, `DumpTraceCompletionStats` searches for a `completionEventCode` event that occurred later in time *and* has the same `aValue` field.

`filename`

A filename to dump the log to.

NOTE: If `subscribertraceutils.c` is compiled to dump the log via `printf` to the debugger Terminal, the `filename` arg is ignored.

Implementation

Streaming library call.

Associated Files

<:streaming:subscribertraceutils.h>, libsubscriber.a

See Also

`AddTrace()`, `DumpRawTraceBuffer()`

Chapter 2

Data Streaming Tools

This section presents the reference documentation for the Data Streaming stream preparation tools.

AudioChunkifier

Chunkifies an AIFF or AIFC sampled audio file.

Format

AudioChunkifier [options] -i <infile> -o <outfile>

Description

Converts a standard AIFF or compressed-AIFF ("AIFC") format sampled audio file to a chunkified file of chunk type 'SNDS'.

Supported audio types

The following table lists the various AIFF and AIFC sampled audio data formats supported by the AudioChunkifier tool, as well as each format's tag ID, DSP instrument name, and data bandwidth.

Data Type Tag ID	DSP Instrument Name	Data Type Description	
Data BW			
-----	-----	-----	
SA_22K_8B_M KB/s	sampler_8_v1.dsp	22.05 kHz 8-bit mono uncompressed	22
SA_22K_8B_S KB/s	sampler_8_v2.dsp	22.05 kHz 8-bit stereo uncompressed	44
SA_22K_16B_M KB/s	sampler_16_v1.dsp	22.05 kHz 16-bit mono uncompressed	44
SA_22K_16B_S KB/s	sampler_16_v2.dsp	22.05 kHz 16-bit stereo uncompressed	88
SA_44K_8B_M KB/s	sampler_8_f1.dsp	44.1 kHz 8-bit mono uncompressed	44
SA_44K_8B_S KB/s	sampler_8_f2.dsp	44.1 kHz 8-bit stereo uncompressed	88
SA_44K_16B_M KB/s	sampler_16_f1.dsp	44.1 kHz 16-bit mono uncompressed	88
SA_44K_16B_S KB/s	sampler_16_f2.dsp	44.1 kHz 16-bit stereo uncompressed	176
SA_22K_16B_M_SQS2 KB/s	sampler_sqs2_v1.dsp	22.05 kHz 16-bit mono compressed 2:1	22

SA_44K_16B_M_SQS2 KB/s	sampler_sqs2_f1.dsp	44.1 kHz 16-bit mono compressed 2:1	44
SA_22K_16B_M_CBD2 KB/s	sampler_cbd2_v1.dsp	22.05 kHz 16-bit mono compressed 2:1	22
SA_22K_16B_S_CBD2 KB/s	sampler_cbd2_v2.dsp	22.05 kHz 16-bit stereo compressed 2:1	44
SA_44K_16B_M_CBD2 KB/s	sampler_cbd2_f1.dsp	44.1 kHz 16-bit mono compressed 2:1	44
SA_44K_16B_S_CBD2 KB/s	sampler_cbd2_v2.dsp	44.1 kHz 16-bit stereo compressed 2:1	88
SA_22K_16B_M_ADP4 KB/s	sampler_adp4_v1.dsp	22.05 kHz 16-bit mono compressed 4:1	11
SA_44K_16B_S_ADP4 KB/s	sampler_adp4_v1.dsp	44.1 kHz 16-bit mono compressed 4:1	22

Arguments

-cs <chunksize>

Size in bytes of output SNDS data chunks. Only numeric characters are allowed. (Good: 4096. Bad: 4K. Bad: 4,096.) Default: 16384

-i <infile>

Input AIFF or AIFC file name.

-ia <amp>

Initial amplitude. Can be changed with a control message while the stream is playing. Default: 0x7500

-ip <pan>

Initial pan. Can be changed with a control message while the stream is playing. Default: 0x4000

-lc <chan>

Specifies the logical channel number of the chunkified output file. The Audio Subscriber supports up to eight logical channels per stream. Audio streams on different logical channels can be of different AIFF or AIFC audio file formats. Each channel maintains its own amplitude and pan values. Generally one of these channels is considered the Oclock channel and the timestamps in its data chunks drive the stream clock. 0 is the default clock channel. The clock channel can be set while the stream is playing with an SAudioSubscriber control message. Default: 0

-nb <num>

Maximum number of SNDS chunks that can be submitted to the audio folio at any one time. Any additional SNDS chunks are queued up separately and submitted to the folio as previously submitted buffers complete. Default: 8

-o <outfile>

Output chunkified audio file name.

Caveats

In order to play an audio-only stream correctly, the chunkified audio file **MUST** be processed by the Weaver.

Example

```
AudioChunkifier -cs 4096 -i soundfile.aiff -o soundfile.snds
```

SAudioTool

Prints or modifies header information of chunkified AIFF files.

Format

SAudioTool [options] -i <infile>

Description

SAudioTool can perform either of two functions: (1) print out the header information (see "Examples" section below) of chunkified audio files created by AudioChunkifier, or (2) modify the header information (and data chunks, if necessary).

Arguments

To dump out the existing header info without modification, use only the -dh and -i arguments. Using any of the other arguments will modify the input file.

-dh

Dump header data to screen.

-l <chan>

Specify logical channel. Modifies input file.

-nb <num>

Specify umber of audio folio buffers to use. Modifies input file.

-ia <amp>

Specify initial amplitude (in hex). Modifies input file.

-ip <pan>

Specify initial pan (in hex). Modifies input file.

-i <infile>

Input chunkified audio file name.

Examples

```
SAudioTool -dh -i soundfile.snds
```

In the example above, SAudioTool prints out the following info:

```
logical channel
number of audio folio buffers
initial amplitude
initial pan
sample size
sample rate
number of channels
compression type
compression ratio
total number of samples
```

```
SAudioTool -lc 1 -ia 0x4000 -i soundfile.snds
```

In the example above, SAudioTool modifies the logical channel number and the input amplitude of the chunkified input file.

DATAChunkify

Chunkifies a data file for the DATA subscriber.

Format

```
DATAChunkify <arg>*
  -? or -help      print usage info.
  -i <file>         input file name [REQUIRED].
  -o <file>         output file name [REQUIRED].
  -chan <num>       channel number (default = 0).
  -t <ticks>        first chunk's time in audio folio ticks
(0).
  -to <ticks>       time offset for subsequent chunks (0).
  -cs <size>        size of each data chunk in bytes
(16384).
  -m <num>          memtype bits (0).
  -u <type><type>    user data word (0000000000000000).
  -comp <type>      compressor (NONE).
Valid compressors are:
  NONE    No compression.
  3DOC    3DO compession folio compression.
```

Description

This tool breaks files into chunks suitable for the DATA subscriber. The files output by this tool must be woven into a stream file with the Weaver tool in order to be used by the DataStreamer.

EZFlixChunkifier

Chunkifies an uncompressed QuickTime movie.

Format

```
EZFlixChunkifier [options] <moviefile>
```

Description

Converts an uncompressed ("raw") QuickTime movie to a 3DO chunkified format. Performs EZFlix compression as part of this conversion. Is an alternative to compressing with a QuickTime app using the EZFlix QuickTime component and then chunkifying with QTVideoChunkifier. Enjoys sentences without subjects.

Arguments

- c <channel>
Logical channel number. Default: 0
- f <framerate>
Specifies the frame rate in fps. Default: 15
- h <frameheight>
Specifies the maximum image frame height, in pixels, which will be encoded. If the actual height of the input QuickTime movie's frames is greater than <frameheight>, then the tool will crop vertically to this number. Default: 192
- i <inputfilename>
Specifies the QuickTime movie input filename.
- o <outputfilename>
Specifies the output filename. If no name is specified, the output filename will be the input filename with .EZFL appended to the end.
- opera
Output an Opera format stream.
- q
Specifies the EZFlix video compression quality, on a scale from 0 to 100. Currently supported values include 10, 25, 40, 60, or 75. Default: 25
- v
Enables verbose diagnostic output.
- w <framewidth>
Specifies the maximum image frame width, in pixels, which will be encoded. If the actual width of the input QuickTime movie's frames is greater than <framewidth>, then the tool will crop horizontally to this number. Default: 256

Example

```
EZFlixChunkifier -i video.moov -o video.EZFL -w 288 -h 208 -f 12
```

QTVideoChunkifier

Chunkifies an EZFlix-compressed QuickTime movie.

Format

```
QTVideoChunkifier [options] <moviefile>
```

Description

Converts a EZFlix-compressed QuickTime movie to a 3DO chunkified format. For convenience, creates a video-only stream that can be played by the Data Streamer without weaving (though typically there will be reason to weave eventually, to add audio, etc.)

Arguments

-b <size>

Stream block size in bytes of the output video-only stream. Only meaningful if the video-only stream is to be played as is. If later processed by the Weaver, will be over-ridden by the Weaver-specified stream blocksize. Must be a multiple of 2048 bytes. Default: 32768

-c <channel>

Logical channel number. Default: 0

-d <outputDirectory>

Direct the output files to a directory.

-l

Causes a GOTO chunk to be output at the end of the stream, which will make the streamed movie loop back to its first frame. NOTE: This feature is currently disabled.

-m <rate>

Specifies the rate, in markers per second, at which key-frame markers will be output. (All EZFlix frames are key frames.) Default: 1

-o <outputfilename>

Specifies the output filename. If no name is specified, the output filename will be the input filename with .EZFL appended to the end.

-opera

Output an Opera format stream.

-s

Specifies the start time.

-v

Enables verbose diagnostic output.

Caveats

QTVideoChunkifier ignores the input file's audio and outputs video chunks only.

Example

```
QTVideoChunkifier video.MooV
```

MPEGAudioChunkifier

Chunkifies an MPEG-1 Audio bitstream file.

Format

MPEGAudioChunkifier [-i <inputFile> -o <outputFile> [-s <startTime>]]

Description

This MPW tool chunkifies an MPEG-1 audio bitstream file. The MPEG-1 audio file **MUST** be encoded using 44.1khz sampling rate, layer 2 encoding, fixed bitrate (not free format), and must in all other ways conform to the MPEG-1 audio bitstream syntax.

Arguments

If no arguments are given, the MPEGAudioChunkifier will prompt for all paramters.

-i <inputFile>

Specifies the input MPEG-1 audio bitstream file to be chunkified.

-o <outputFile>

Specifies the output file to be created or overwritten. No warning is given if the file exists.

-s <startTime>

Specifies the chunkified stream's start time in audio ticks. If not specified, defaults to 0.

Examples

```
MPEGAudioChunkifier -i music.mpa -o music.mpa.chunk -s 240
```

Chunkifies the file "music.mpa" and places the result in the file "music.mpa.chunk". The start time of the chunkified file is 240 audio ticks (approximately 1 second).

```
MPEGAudioChunkifier -i voice.mpa -o voice.mpa.chunk
```

Chunkifies the file "voice.mpa" and places the result in the file "voice.mpa.chunk". The start time of the chunkified file is 0 audio ticks.

```
MPEGAudioChunkifier
```

With no arguments, the MPEGAudioChunkifier will prompt for the input file, output file, and start time.

MPEGVideoChunkifier

Chunkifies an MPEG-1 Video elementary stream.

Format

```
MPEGVideoChunkifier (-prompt) |  
(-i <infile> -o <outfile> -d <time> -s <time>) [options]
```

Description

Converts an MPEG-1 Video "elementary stream" (an ISO standard MPEG-1 video-only compressed data stream) to a chunkified MPEG-1 video format. As an alternative to the command-line interface, a prompt mode is available by typing 'MPEGVideoChunkifier -prompt'. Typing the command with no arguments prints out usage, help and examples.

For the -d and -s parameters, time is specified in M2 audio ticks, that is, (44100/184) or approximately 240 ticks per second. Thus, for a 24 fps second movie, frames come at intervals of approx. 10 audio ticks. Frame intervals for a 30 fps movie are at approx 8 audio ticks.

Arguments

- prompt
Puts MPEGVideoChunkifier into user-prompt mode. In this mode user will be prompted for all other arguments.
- i <infile>
File name of the input MPEG-1 Video elementary stream.
- o <outfile>
File name of the output MPEG-1 Video chunkified stream.
- d <time>
Delay in audio ticks between stream time (aka "decoding time" or "delivery time") and presentation time.
- s <time>
Presentation time in audio ticks of first displayed frame.
- p
Progress report (or "verbose" mode). A very useful feature because it prints out the stream time (or "decoding" time), the presentation time, and the (I, B, or P) frame type of each video frame. Note that decoding time should be used for branching, since time-based markers and GOTO chunks use stream time.
- fps <framerate>
Frame rate in fps. Use this argument only if it's necessary to override the "picture_rate" parameter specified in video sequence layer of the input MPEG Video elementary stream.

Caveats

MPEGVideoChunkifier does not support ISO MPEG-1 Systems streams, which is an ISO-defined format for multiplexed MPEG-1 Video and MPEG-1 Audio streams. If you wish to chunkify the MPEG-1 Video part of an MPEG-1 Systems stream, you must use first demultiplex out the video. There are a number of public-domain MPEG-1 Systems "splitter" tools which will do this.

Examples

MPEGVideoChunkifier -prompt

MPEGVideoChunkifier -i mymovie.mpg -o mymovie.MPVD -d 10 -s 10

DumpStream

Writes diagnostic listing of a stream.

Format

DumpStream [options] -i <inputfile>

Description

DumpStream writes a diagnostic listing of stream files output by the Weaver and chunkified files output by the various chunkifier. This can help greatly with debugging and complex stream-file creation.

Arguments

- i <inputfile>
Input stream file name.
- s <time>
Stream time in audio ticks at which dump begins.
- bw <bps>
Maximum data bandwidth.
- v
Enable verbose output.
- xf
Outputs file positions (byte offsets) in hex rather than decimal.
- bs
Stream block size.
- w
Suppress warnings.
- stats
Statistics output only.
- fbs
Filler block sizes output only.

Example

```
DumpStream -xf -v -i video1.stream
```

Weaver

Multiplexes chunkified streams into one playable stream.

Format

```
Weaver [options] -o output_file < script_file
```

Description

The Weaver takes as input a script file that references one or more chunked files and writes one stream that consists of interleaved chunks from all input files. The Weaver script specifies how the weaving is done. (See the chapter or on-line help on the topic "Weaver Script Commands" which begins with audioclockchan.)

Weaver scripts consist of commands and comments. Commands start with a Weaver command followed by command arguments. Comments are lines starting with a #.

Arguments

-iobs <size>

Mac buffer size in bytes used by Weaver. Default: 32768

-b <time>

Output stream start time in audio ticks. May also be specified by the streamstarttime Weaver-script command. Default: 0

-s <size>

Output stream block size. May also be specified by the streamblocksize Weaver-script command.

-m <size>

Output stream media block size. May also be specified by the mediablocksize Weaver-script command.

-o <file>

Output stream file name.

-mm <num>

Max markers to allow.

-v

Enable verbose diagnostic output.

Caveats

Make sure to quote file or folder names that contain spaces. Use writestreamheader in every Weaver script.

Example

```
set WeaveScript ¶
'writestreamheader                                ¶n¶
streamblocksize      32768                        ¶n¶
streambuffers        4                            ¶n¶
streamerdeltapri     -10                          ¶n¶
dataacqdeltapri      -9                           ¶n¶
subscriber           MPVD -2                      ¶n¶
```

```
subscriber          SNDS 11          ¶n¶
preloadinstrument    SA_44K_16B_S_CBD2 ¶n¶
file SF.MPVD         1 0             ¶n¶
file raggae.SNDS     0 24            ¶n'
echo "{WeaveScript}" | Weaver -o SF.stream
```

audioclockchan

Specifies which audio subscriber channel will drive the stream clock.

Synopsis

```
audioclockchan <audio_clock_channel>
```

Description

Specifies which audio subscriber channel will drive the data stream presentation clock. The default is audio channel 0.

This information will be useful IF the stream has audio data on the specified channel.

The audio subscriber will adjust the Data Streamer's presentation clock to match the time stamps in data chunks that arrive for this channel. If a stream has audio data throughout, using this will help synchronize audio with video, esp. when the stream starts up and when it branches.

Arguments

audio_clock_channel

An integer which specifies which audio channel will drive the Data Streamer's clock.

Legal channel numbers are from 0 to 31.

Caveats

This command sets a stream header chunk field. The `writestreamheader` command must also be in the weave script for this to have any effect.

Example

```
audioclockchan 1
```

See Also

Weaver, `writestreamheader`; `DSHeaderChunk` in `<:streaming:dsstreamdefs.h>`

dataacqdeltapri

Specifies the relative priority for the Data Acquisition thread.

Synopsis

```
dataacqdeltapri <delta_priority>
```

Description

Specifies the priority of the Data Acquisition thread relative to the priority of the client application of the Data Streaming library.

Arguments

`delta_priority`

An integer to add to the application thread's priority to compute the Data Acquisition thread's priority. A positive value will make the Data Acq thread higher priority than the app, while a negative value will make it lower.

Caveats

The computed priority must be in the range 11 to 199.

This command sets a stream header chunk field. The `writestreamheader` command must also be in the weave script for this to have any effect.

Example

```
dataacqdeltapri -9
```

See Also

Weaver, `streamdeltapri`, `writestreamheader`, `DSHeaderChunk` in `<:streaming:dsstreamdefs.h>`

enableaudiomask

Specifies which audio channels to pre-enable.

Synopsis

```
enableaudiomask <bit_mask>
```

Description

Specifies which audio channels to pre-enable before stream playback begins. Legal channel numbers are from 0 to 31.

The application can also enable and disable channels by calling `DSSetChannel()`.

Arguments

`bit_mask`

A hex bit mask which specifies which audio channels will be enabled, as indicated by the position of bits set to 1 in a 32-bit register. Thus, value 0x3 enables audio channels 0 and 1.

Caveats

This command sets a stream header chunk field. The `writestreamheader` command must also be in the weave script for this to have any effect.

Example

```
enableaudiomask 0x3
```

See Also

Weaver, `writestreamheader`; `DSHeaderChunk` in `<:streaming:dstreamdefs.h>`

file

Weaves an input stream file into the woven stream.

Synopsis

```
file <input_file> <priority> <time_offset>
```

Description

Specifies an input stream file to "weave" (multiplex) with other input stream files. The priority determines the order in which any chunks with identical timestamps are written to the output stream. Smaller priority numbers are higher priority. The time offset is similar to the `streamstarttime` offset except it only offsets this input file's chunks.

Arguments**input_file**

Name of a chunkified file to weave.

priority

An integer which determines the relative order, compared to other input files, to output of chunks into the woven stream when two or more chunks from different streams have identical timestamps. Value 0 is the highest priority.

time_offset

Time, in audio ticks, to offset this input file's chunks when writing them to the woven output stream.

Example

```
file raggae.SNDS 0 24
```

See Also`Weaver`, `streamstarttime`

markertime

Adds an entry to the marker table to use as a branch destination.

Synopsis

```
markertime <time>
```

Description

A stream file can contain a marker table: a table of branch destination points in that file. Each marker entry is a <time, offset> pair which allows the streamer to branch to the given byte offset within the file and set the stream clock to the given presentation time.

A call to `DSGoMarker()` or a `STRM GOTO` chunk (see `writetogochunk`) can branch to the *n*th marker in the marker table. `DSGoMarker()` can also branch to an absolute byte offset within the file (e.g. 0) (which doesn't require a marker table), or to the marker with a specified time, or other alternatives.

The `markertime` command asks the Weaver to add an entry to the marker table that it's building up. You specify the destination time. The Weaver will find the matching byte offset.

The `markertime` command also asks the Weaver to place all pre-branch data (that is, all data chunks with timestamps before the marker time) before the marker point, and to place all post-branch data (data chunks with timestamps at or beyond the marker time) after the marker point.

The `markertime` command also asks the Weaver to place the post-branch data at the start of a stream block even if that requires inserting a `FILL` chunk to fill out the current stream block. (In the future, there could be an "unfilledmarkertime" command. An unfilled marker conserves stream space and bandwidth but costs seek covering time since the streamer will skip part of the first block read after branching.)

Include a `markertime` command in your weave script for each location you wish to branch to.

Arguments

time

An integer which specifies the marker's stream time position in audio ticks. Allowed values are integers in the range 0 to 7FFFFFFF (hex).

Caveats

This command adds an entry to the marker table. The `writemarkertable` command must also be in the weave script for this to have any effect.

Example

```
markertime 800
markertime 1200
writemarkertable
```

See Also

Weaver, `writemarkertable`, `writetogochunk`; `DSMarkerChunk` in
<:streaming:dsstreamdefs.h>

mediablocksize

Specifies the block size of the media. (Use 2048 for CD-ROM.)

Synopsis

```
mediablocksize <blocksize>
```

Description

Mediablocksize specifies the blocksize of the media--the run-time source of the stream file. This value can also be specified by the "-m" Weaver command-line argument.

If this command is not present in the weave script, the default value is 2048 bytes (the blocksize of most CD-ROM drives).

What the Weaver does with the media blocksize info is cross-check that it evenly divides the stream block size, as specified by the streamblocksize weave script command or the "-s" Weaver command-line argument. This is important because the Data Streamer always reads the stream file in units of stream blocks via block-oriented file I/O. If the stream block isn't an integral number of media blocks, the streamer will fail with I/O errors.

Arguments

blocksize

An integer no greater than 7FFFFFFF (hex).

Example

```
mediablocksize 2048
```

See Also

Weaver, streamblocksize

numsubsmessages

Specifies number of subscriber messages for the Data Streamer to allocate.

Synopsis

```
numsubsmessages <number_of_messages>
```

Description

Specifies the number of subscriber messages that the Data Streamer should allocate to run a given stream.

If this command is not present in the weave script, the default value is 256.

Arguments

`number_of_messages`

An integer specifying the number of subscriber messages to allocate.

Caveats

If the Data Streamer doesn't allocate enough subscriber messages, it will get stuck and abort playback.

This command sets a stream header chunk field. The `writestreamheader` command must also be in the weave script for this to have any effect.

Example

```
numsubsmessages 200
```

See Also

Weaver, `writestreamheader`; `DSHeaderChunk` in `<:streaming:dsstreamdefs.h>`

preloadinstrument

Specifies an audio DSP instrument to preload.

Synopsis

```
preloadinstrument <audio_data_type>
```

Description

Specifies an audio DSP instrument that the SAudioSubscriber should preload, expressed as an SAudioSubscriber tag name. You can repeat this command to specify up to 16 instruments to preload.

The SAudioSubscriber uses DSP instruments to decode audio samples, and it can load these instruments on demand. But it's important to preload instruments if you're using the SAudioSubscriber since loading them requires seeking the CD which disrupts stream playback.

Arguments

audio_data_type

A string which specifies the Data Type Tag ID. There are tags for both uncompressed (AIFF) and compressed (AIFC) sound types. For example, SA_22K_16B_M refers to uncompressed 22.05 kHz 16-bit mono. SA_44K_16B_S_CBD2 refers to 44.1 kHz 16-bit stereo, compressed 2:1. For a complete listing of all AIFF and AIFC sound types, see the documentation for the AudioChunkifier tool.

Caveats

This command stores info in the stream header chunk. The writestreamheader command must also be in the weave script for this to have any effect.

Example

```
preloadinstrument SA_44K_16B_S_CBD2
```

See Also

Weaver, AudioChunkifier, writestreamheader; DSHeaderChunk in
<:streaming:dstreamdefs.h>

streamblocksize

Specifies the stream block size used by the Weaver.

Synopsis

```
streamblocksize <block_size>
```

Description

Specifies the stream block size which will be used by the Weaver and the run-time Data Streamer. This value can also be specified by the "-s" Weaver command-line argument. The Weaver will check that this value is a multiple of the media block size (see `mediablocksize`).

Typical sizes are 32768, 65536, and 98304. Smaller sizes allow faster branching (since it takes less time to read the first post-branch block) while larger sizes allow higher stream bandwidth.

If this command is not present in the weave script, the default `streamblocksize` is 32768 bytes.

Arguments

`blocksize`

A positive integer no larger than 7FFFFFFF (hex).

Caveats

This command sets a stream header chunk field. The `writestreamheader` command must also be in the weave script for this to have any effect.

Example

```
streamblocksize 65536
```

See Also

Weaver, `writestreamheader`, `mediablocksize`; `DSHeaderChunk` in
<:streaming:dsstreamdefs.h>

streambuffers

Specifies number of stream buffers the application should allocate.

Synopsis

```
streambuffers <number_of_buffers>
```

Description

Specifies the number of buffers the application should allocate to playback this stream.

If this command is not present in the weave script, the default number of stream buffers is 4.

For most situations, at least 4 stream buffers are needed to ensure smooth, robust, synchronized streaming. It may be possible to get by with just 3 in some situations.

Arguments

number_of_buffers
A positive integer.

Caveats

This command sets a stream header chunk field. The writestreamheader command must also be in the weave script for this to have any effect.

Example

```
streambuffers 6
```

See Also

Weaver, writestreamheader, streamblocksize; DSHeaderChunk in
<:streaming:dsstreamdefs.h>

streamdeltapri

Specifies the relative priority for the Data Stream thread.

Synopsis

```
streamdeltapri <delta_priority>
```

Description

Specifies the priority of the Data Stream thread relative to the priority of the client application of the Data Streaming library.

Arguments

`delta_priority`

An integer to add to the application thread's priority to compute the Data Stream thread's priority. A positive value will make the Data Stream thread higher priority than the app, while a negative value will make it lower.

Caveats

The computed priority must be in the range 11 to 199.

This command sets a stream header chunk field. The `writestreamheader` command must also be in the weave script for this to have any effect.

Example

```
streamdeltapri -10
```

See Also

Weaver, `dataacqdelta pri`, `writestreamheader`; `DSHeaderChunk` in `<:streaming:dsstreamdefs.h>`

streamstarttime

Specifies a time offset for chunks going into the output stream.

Synopsis

```
streamstarttime <time>
```

Description

Specifies a time offset for chunks going into the output stream. This value can also be specified by the "-b" Weaver command line argument.

If this command is not present in the weave script, the default value is 0.

Arguments

time

A positive integer giving the offset in audio ticks.

Example

```
streamstarttime 240
```

Assuming all input streams start at time zero (which they'd better!), this will add 240 ticks (approximately 1 second) to the time stamp of each chunk when writing it to the woven stream.

See Also

Weaver

subscriber

Asks to instantiate a subscribe for a particular data type.

Synopsis

```
subscriber <subscriber_ID> <delta_priority>
```

Description

Adds a subscriber entry into the stream header which asks the application to instantiate a subscriber for a particular data type. The arguments specify the subscriber data type and relative thread priority (relative to the priority of the client application of the Data Streaming library).

Arguments

subscriber_ID

A 4-byte ASCII code indicating the subscriber data type, e.g.: MPVD for MPEGVideoSubscriber, MPAU for MPEGAudioSubscriber, EZFL for EZFlixSubscriber, SNDS for SAudioSubscriber, and DATA for DataSubscriber.

delta_priority

An integer to add to the application thread's priority to compute the subscriber thread's priority. A positive value will make the subscriber thread higher priority than the app, while a negative value will make it lower. Generally you want to give the audio subscriber the highest priority of all to help audio playback without hiccups.

Caveats

If the run-time program doesn't instantiate a subscriber for some data type, the Data Streamer will ignore all chunks of that data type.

The computed priority must be in the range 11 to 199.

This command stores info in the stream header chunk. The writestreamheader command must also be in the weave script for this to have any effect.

Examples

```
subscriber MPVD -2
subscriber SNDS 11
```

See Also

```
Weaver, writestreamheader; DSHeaderChunk in <:streaming:dsstreamdefs.h>;
NewDataSubscriber(), NewEZFlixSubscriber(), NewMPEGAudioSubscriber(),
NewMPEGVideoSubscriber(), NewSAudioSubscriber()
```

writetogochunk

Writes a STRM GOTO chunk at the specified time in the output stream.

Synopsis

```
writetogochunk <stream_time> <options> <destination>
```

Description

Writes a STRM GOTO chunk at the specified time in the output stream. A GOTO chunk causes stream playback to unconditionally branch when playback reaches the given stream_time.

A GOTO chunk can use any of three alternative ways to specify the branch destination, and can also use some optional flags. The branch alternative and optional flags are encoded in the <options> argument.

The <destination> argument specifies the branch destination point. Its units depend on the <options> branch alternative.

Arguments**stream_time**

Where to place the GOTO chunk, in audio ticks. At runtime, the streamer will begin branching after it delivers chunks that precede the GOTO chunk (chunks with times less than this stream_time).

options

A GOTO chunk can use any of three alternative ways to specify the branch destination, and can also use some optional flags.

branch option 0 = GOTO_OPTIONS_ABSOLUTE, which means branch to an absolute (byte position, time). This alternative doesn't require a marker. NOTE: ABSOLUTE GOTO CHUNKS ARE NOT YET SUPPORTED BY THE WEAVER. A future Weaver could create absolute GOTO chunks. You would specify the branch destination time and the Weaver would compute the branch destination byte position in the same way that it computes marker table entries (see markertime).

branch option 1 = GOTO_OPTIONS_MARKER, which means branch to a marker by marker number, an index into the marker table (see markertime).

branch option 2 = GOTO_OPTIONS_PROGRAMMED, which means do a programmed branch number, i.e. the branch destination is determined by indexing into the programmed branch table. NOTE: PROGRAMMED BRANCHES ARE NOT YET SUPPORTED BY THE WEAVER OR THE DATA STREAMER.

You can also BITOR these branch option flags into the <options> argument:

The flag 1<<8 = GOTO_OPTIONS_FLUSH means flush subscribers instead of doing a butt-joint branch. This would be weird in a GOTO chunk since it means not playing out all the data that precedes the GOTO chunk before branching.

The flag 1<<9 = GOTO_OPTIONS_WAIT means branch then wait for DStartStream. This flag IS NOT YET SUPPORTED BY THE DATA STREAMER.

destination

This specifies where to branch to. Its units depend on the <options> branch alternative.

For `GOTO_OPTIONS_MARKER`, the `<destination>` argument specifies a marker number, that is, an index into the marker table. Use the `markertime` command to generate marker table entries.

For `GOTO_OPTIONS_ABSOLUTE`, the `<destination>` argument specifies a destination stream time, as with `markertime`.

For `GOTO_OPTIONS_PROGRAMMED`, the `<destination>` argument specifies a programmed branch number, which is not yet supported by the Data Streamer.

Example

```
writetogotochunk 3351 1 1
```

See Also

Weaver, `markertime`; `struct StreamGoToChunk` and `enum GOTO_Options` in `<:streaming:dsstreamdefs.h>`

writemarkertable

Writes the marker table to the woven output stream.

Synopsis

```
writemarkertable
```

Description

Asks the Weaver to write a marker table chunk to the woven output stream. You need to use this command in every weave script that uses the `markertime` command.

Arguments

none.

Example

```
markertime 800  
markertime 1200  
writemarkertable
```

See Also

Weaver "-mm" command line argument, `markertime`, `writegotochunk`; `DSMarkerChunk` in [*<:streaming:dsstreamdefs.h>*](#)

writestopchunk

Writes a STRM STOP chunk at the specified time in the output stream.

Synopsis

```
writestopchunk <stream_time>
```

Description

Writes a STRM STOP chunk at the specified time in the output stream.

A STOP chunk causes stream playback to stop much as if the client application called `DSStopStream()`. If the client application has used `DSWaitEndOfStream()` to register for end-of-stream-playback notifications, the streamer will reply to that message with the Err code `kDSSTOPChunk`. The client application can resume playback by calling `DSStartStream()`, presuming after re-registering for end-of-stream-playback notifications.

Arguments

`stream_time`

Where to place the STOP chunk, in audio ticks. At runtime, the streamer will STOP after it finishes playing chunks that precede this STOP chunk (chunks with times less than this `stream_time`).

Example

```
writestopchunk 2400
```

See Also

Weaver, struct `StreamStopChunk` in `<:streaming:dsstreamdefs.h>`, `<:streaming:datastreamlib.h>`.

writestreamheader

Writes the stream header to the woven output stream.

Synopsis

```
writestreamheader
```

Description

Asks the Weaver to write a stream header chunk to the woven output stream. The values in the stream header chunk are supplied by other weave script commands and by Weaver command line arguments.

Arguments

none.

Caveats

You should write a stream header in every woven stream.

Example

```
writestreamheader
```

See Also

Weaver, audioclockchan, dataacqdelta pri, enableaudiomask, numsubsmessages, preloadinstrument, streamblocksize, streambuffers, streamdelta pri, subscriber; DSHeaderChunk in *<:streaming:dstreamdefs.h>*



3DO M2 Command List Toolkit

Version 2.0 – May 1996

Copyright © 1996 The 3DO Company and its licensors.

All rights reserved. This material constitutes confidential and proprietary information of The 3DO Company. This documentation is subject to a license agreement with The 3DO Company and may be used only by parties to such agreement. Use by any other persons, and/or for any purpose not expressly authorized by the agreement, is strictly prohibited.

3DO's LICENSOR(S) MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. 3DO'S LICENSOR(S) DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME JURISDICTIONS. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

Other brand or product names are the trademarks or registered trademarks of their respective holders.

Contents

1

The Command List Toolkit

Introduction.....	CLT-2
Vertex Instructions	CLT-5
State Control Instructions.....	CLT-7
Flow Control Instructions	CLT-8
Command List Snippets	CLT-10

2

Graphics State (GState)

Introduction.....	CLT-12
Creating and Initializing a GState.....	CLT-12
Using a GState.....	CLT-13
Synchronizing the Triangle Engine to Video Signals through GState.....	CLT-14
Cleanup Routines	CLT-15
Combining CLT, the 2D and 3D Pipelines and Frameworks, and the Font Folio.....	CLT-15

3

Textures

Introduction.....	CLT-18
Texture Mapping	CLT-19
Using Filtering in Texture-Mapping Operations	CLT-19
Replicating Textures	CLT-19
Levels of Detail (LODs)	CLT-20
Using Mipmaps	CLT-21
Mipmap Filtering	CLT-21

How M2 Mipmap Filtering Works	CLT-22
Using Mipmap Filtering.....	CLT-24
Nearest (Point) Filtering	CLT-24
Linear Filtering	CLT-24
Bi-linear Filtering	CLT-26
Quasi Tri-linear Filtering	CLT-26
Perspective Correction.....	CLT-27
Computing the U and V Coordinates for Texture Mapping	CLT-27
Using Perspective Off Mode in 2D Operations	CLT-27
How Textures Are Stored in Memory.....	CLT-28
Texture Formats.....	CLT-29
How Texels Are Stored in Memory	CLT-29
How Uncompressed Texels are Stored	CLT-30
How Compressed Texels are Stored	CLT-30
Special Settings in the Control Byte	CLT-31
PIP Tables	CLT-32
How M2 Interprets Texel Data	CLT-33
How PIP Tables Work	CLT-33
Texture Application Blending	CLT-34
Multiplying Color and Alpha Components	CLT-34
Application Modes	CLT-35
Texture Mapping Step-by-Step	CLT-35
Loading Textures	CLT-36
Loading Textures by Using MMDMA	CLT-37
Loading Uncompressed Textures	CLT-37
Compressed Texture Load	CLT-39
Texture Mapper Pipeline Register Definitions	CLT-40
Command Registers, Fields, and Macros	CLT-40
Texture Generation Registers	CLT-42
Texture Loader Register Definitions	CLT-51

4

Destination Blender

Pixel Discards	CLT-60
Pixel Blending	CLT-60
Pixel Scaling	CLT-61
Source Blending	CLT-61
Dithering.....	CLT-62
Z-buffering.....	CLT-62

Z-banding	CLT-62
Window Clipping	CLT-63
Destination Blender Register Definitions	CLT-63
Register Memory Map	CLT-77

5

Register Setups

Z-Buffering	CLT-81
Dithering.....	CLT-82
Setting Up the Destination Blender Source Buffer	CLT-82
Transparencies	CLT-82
Texturing	CLT-83
Perspective.....	CLT-84
Sprites	CLT-84
Lighting	CLT-84
Tiling a Texture.....	CLT-86
Load an Uncompressed Texture	CLT-86
Loading a PIP Table.....	CLT-86

6

Sample Code

7

Function Calls

List of Figures

Figure 1-1 Triangle Engine data flow	CLT-2
Figure 1-2 Texture Mapper data flow	CLT-3
Figure 1-3 Destination Blender data flow	CLT-4
Figure 1-4 Triangle strip	CLT-6
Figure 3-1 Texture Coordinates	CLT-18
Figure 3-2 The texture-mapping process	CLT-19
Figure 3-3 Mipmaps	CLT-20
Figure 3-4 Tradeoffs in filtering operations	CLT-23
Figure 3-5 Bi-linear filtering	CLT-26
Figure 3-6 Storage format of an uncompressed texel	CLT-30
Figure 3-7 Storage format of a ompressed texel	CLT-30
Figure 3-8 Bits in the control byte	CLT-30
Figure 3-9 How textures and texels work with PIP tables	CLT-32
Figure 3-10 Rendering a ghost	CLT-33
Figure 3-11 Bit packing of a Color	CLT-34
Figure 3-12 Placing Load Rectangles inside a texture	CLT-36
Figure 3-13 An array of uncompressed texels ready for loading	CLT-37
Figure 3-14 Uncompressed Loader Setup	CLT-38
Figure 3-15 TxtLdSrcType01	CLT-52
Figure 3-16 TxtLdSrcType23	CLT-53
Figure 3-17 TxtExpType	CLT-53
Figure E-1 Dithering matrix	CLT-82

List of Tables

Table 3-1 Texture Filtering Modes	CLT-22
Table 3-2 Registers for specifying LOD Regions	CLT-24
Table 3-3 Texel Types.....	CLT-29
Table 3-4 Color, Alpha, and SSB Values that Can Be Assigned to a Texel	CLT-29
Table 3-5 Register Memory Map	CLT-41

Preface

About This Book

This book describes the Command List Toolkit and its relationship to the M2's 3D hardware rendering engine.

About the Audience

This manual is written for title developers. The information presented and the level of description is based on the assumption that you are familiar with both the C programming language, and three-dimensional graphics.

How The Command List Toolkit Programmer's Guide Is Organized

Chapter 1, The Command List Toolkit -Provides a brief overview of the M2 3D hardware rendering engine, an introduction to the Command List Toolkit, and descriptions of the associated macros.

Chapter 2, The Graphics State (GState) -Describes the GState object and how it manages the command list buffers.

Chapter 3, Textures -Provides an overview of the M2's texture mapping logic, as well as descriptions of all of the applicable registers.

Chapter 4, Destination Blender -Provides an overview of the M2's destination blending logic, as well as descriptions of all of the applicable registers.

Appendix A, Register Setups -Provides the register setups for a variety of operations.

Appendix B, Sample Code -Provides a sample program.

Appendix C, Function Calls -Describes each of the Command List Toolkit function calls.

Related Documentation

The following documents are recommended as a source of additional information:

- ◆ Foley, van Dam, Feiner, and Hughes, *Computer Graphics Principles and Practice*, Addison-Wesley, 1990
- ◆ 3DO M2 Graphics Programmer's Guide

The Command List Toolkit

The M2's 3D hardware rendering engine (the Triangle Engine or TE) renders graphics by reading blocks of data, called *command lists*, from memory and executing the instructions found in these blocks. The Command List Toolkit (CLT) is a collection of routines and macros that allow you to build these command lists easily and efficiently. Applications use the CLT to build commands for the Triangle Engine. These commands are built up into command list buffers, then handed off to the Triangle Engine's device driver.

The TE executes the commands, found in the command list buffers, asynchronously. So, while the Triangle Engine works on one block of command lists, the CPU is free to begin work on preparing the next set of Triangle Engine commands.

The main job of the CLT is to assign a set of names to all of the Triangle Engine registers and commands. These names are provided to the user through a header file. In addition, several shortcuts for using these names are provided in the form of C preprocessor macros.

This chapter describes the macros provided by the CLT. Your application can use these macros to communicate, at the most rudimentary level, with the Triangle Engine.

This chapter contains the following topics:

Topic	Page Number
Flow Control Instructions	8
Vertex Instructions	8
State Control Instructions	8

Topic	Page Number
Flow Control Instructions	8
Command List Snippets	8

Introduction

Before we talk about the Command List Toolkit, let's take a brief look at how the Triangle Engine works. The TE is primarily composed of two groups of logic: the Texture Mapper and the Destination Blender. illustrates the data flow through the Triangle Engine. See Chapters 3 and 4 for more information about the Texture Mapper and the Destination Blender.

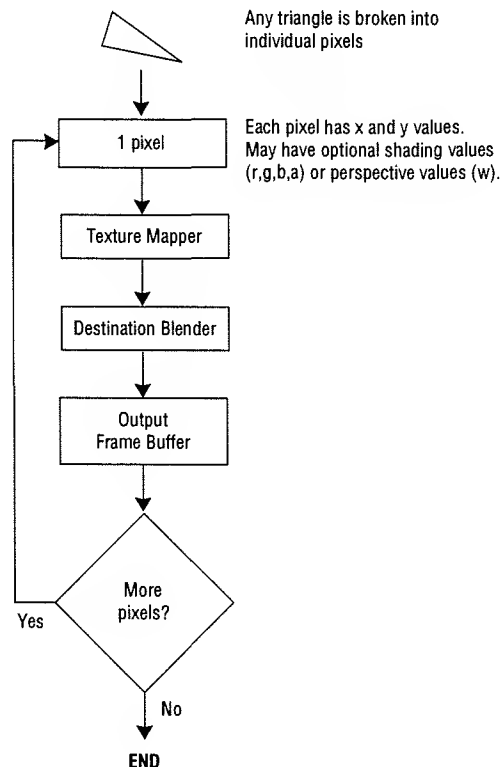


Figure 1-1 Triangle Engine data flow.

The TE renders triangles by first breaking them down into individual pixels. Each pixel has a set of *xy* coordinates that identify it, and may have optional values that define its shading or perspective.

The pixel goes first to the Texture Mapper. The texture mapper logic can render portions of a 2D image onto a 3D triangle, and it is here where texture and color are applied to the pixel (see Figure 1-2).

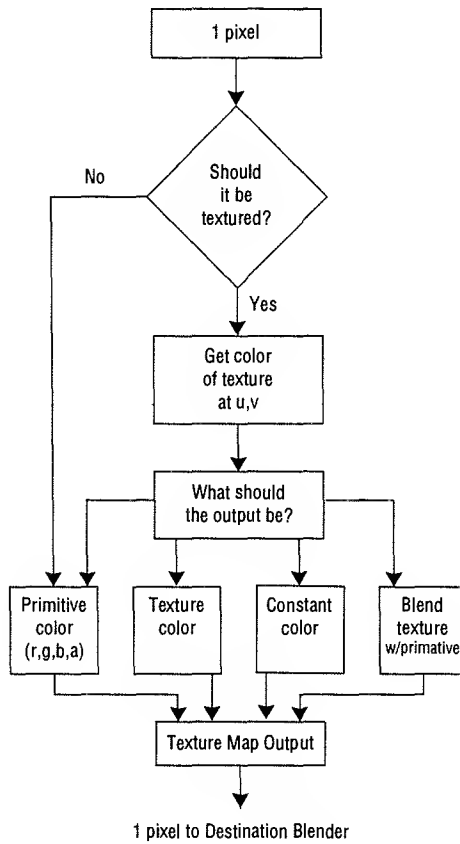


Figure 1-2 *Texture Mapper data flow.*

Next stop is the Destination Blender. The destination blender logic takes the pixel's texture information, and combines (blends) it with data from a number of sources. These sources can include the current Source frame buffer, a variety of constants and the control registers (see Figure 1-3).

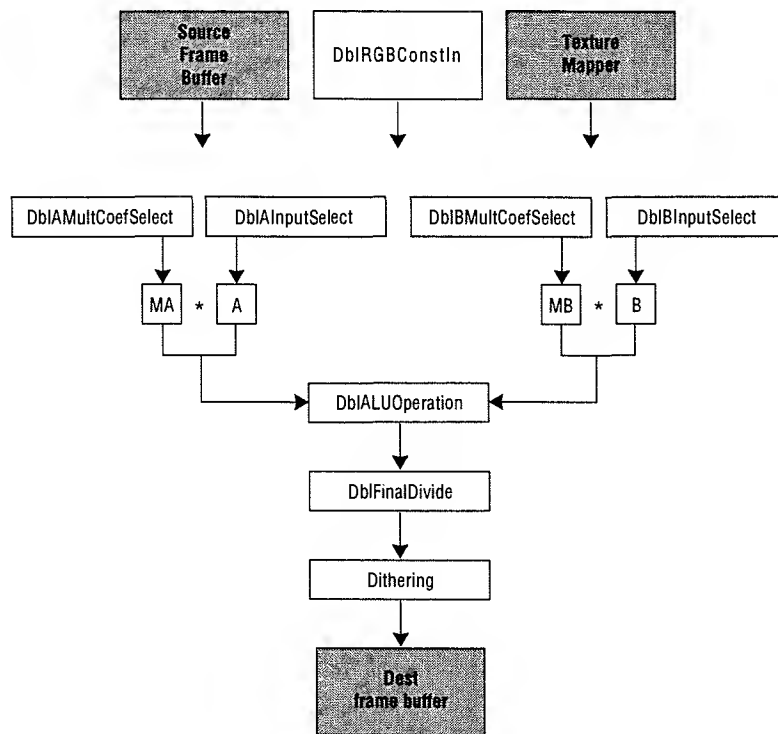


Figure 1-3 Destination Blender data flow.

The Triangle Engine works in a *deferred mode*, meaning that it executes commands from a buffer, rather than responding immediately to each of the commands that you issue. The normal flow of these commands usually looks similar to the list below:

1. Provide information about texture's format, location in memory, etc.
2. Load the texture.
3. Provide information about the texture's PIP (Pen Index Palette, or color table).
4. Load the PIP.
5. Set up the other hardware registers, such as the Destination Blend unit.
6. Provide vertices of one or more triangle strips or fans, to be rendered using the parameters described in Steps 1 - 5, above.
7. Issue a SYNC instruction. Refer to "Flow Control Instructions," later in this chapter, for more information about the SYNC instruction.
8. Return to Step 1.

The only step in the process that actually causes pixels to be rendered is Step 6. The other steps describe how those pixels will be rendered. Note that the SYNC instruction is needed after a collection of vertex instructions have been issued to the Triangle Engine, but it is described in more detail later.

Vertex Instructions

Since vertex instructions are what actually cause pixels to be rendered, we'll start by describing the CLT macros that create these instructions. For now, let's assume that the Texture Mapper, Destination Blender, and the other setup portions of the Triangle Engine are all set to allow these instructions to render correctly.

The general form of vertex instructions is one header word, followed by three or more vertices. The header word describes what the format of the vertices will be, a count of the vertices that follow, and also tells the Triangle Engine whether the vertices describe a triangle strip, or a triangle fan. Vertices contain x and y data, and may also contain RGBA color data, uv texture mapping data, and a $1/w$ perspective correction value (also used for depth buffering). The actual contents of a vertex instruction depends on the macro you choose to use.

The macro to create a vertex header instruction is of the form:

```
CLT_TRIANGLE (pp, stripfan, perspective, texture, shading, count)
```

`pp` – the address of a pointer to the current insertion point in a command list buffer. See the section on GState for more details about this command list buffer pointer. The address of the pointer is necessary, because CLT updates the pointer as it adds instructions to the command list buffer.

Note: See Chapter 2 for more details about the command list buffer pointer.

`stripfan` – an integer stating whether the vertices that follow should be interpreted as a triangle strip or a triangle fan. The constants `RC_STRIP` and `RC_FAN` should be used for this parameter.

`perspective` – a bit stating whether the vertices that follow contain a $1/w$ perspective correction value.

`texture` – a bit stating whether the vertices that follow contain U and V texture coordinates.

`shading` – a bit stating whether the vertices that follow contain R, G, B, and A (alpha) color data.

`count` – an integer count of how many vertices follow this header word. It is important to note that the Triangle Engine can work more efficiently on longer strips and fans than on shorter ones. If optimal rendering speed is desired, the average number of triangles per strip or fan should be 10 or more.

There are several forms of the CLT vertex macros. Each form covers one of the various triangle formats described by the header word. Some examples follow:

```
CLT_Vertex(pp, x, y)
```

```
CLT_VertexRgbaW(pp, x, y, r, g, b, a, w)
```

```
CLT_VertexRgbaUvW(pp, x, y, r, g, b, a, u, v, w)
```

`pp` – the address of a pointer to the current insertion point in a command list buffer. As with all CLT macros that take `pp`, it is updated as data is written to the buffer.

All of the remaining values for the `CLT_Vertex` instructions are represented as IEEE 754 standard 32-bit floating point numbers.

x , y – the x and y coordinates of the vertex of a triangle. They should be in the range of 0 to bitmap width, and 0 to bitmap height, respectively. Negative numbers are not allowed for x and y , so care has to be taken to clip triangles against the top and left edges of a bitmap. Triangles that extend beyond the right or bottom edges of a bitmap are clipped by the Triangle Engine, if they are within the range of 0.0 to 2048.0.

r , g , b , a – represent the red, green, blue, and alpha components for shading a triangle. This data might be different at each vertex, to allow for Gouraud shading of the triangles, or it can be the same at each vertex, in order to flat shade the triangles. The values should be in the range of 0.0 to 1.0 for each of these components.

u , v – represent the texture coordinates that should be applied to a triangle. The values should be in the range of 0 to texture width, and 0 to 1024, respectively. When perspective correction of the textures is desired, u and v should actually be sent to the Triangle Engine as u/w and v/w . Some textures can be tiled, meaning that its texel data repeats in the x and/or y directions. Note that if u and v exceed the width of the texture, the clamping mask is applied.

$1/w$ – a perspective correction factor. As a result, the range for the w value is $[0,1)$. When $w = 0.0$, a triangle is represented as being infinitely far away. As w approaches 1.0, the perspective correction applied to it becomes less and less, until eventually there is no correction done on the triangles.

Below is a fragment of code that adds two triangle strips to a command list buffer. Again, this code assumes that the texture mapper and destination blender have been correctly set up, and that the variable `listPtr` is a valid pointer to the current insertion point in a command list buffer.

```
{
    ...
    CLT_Triangle( &listPtr, RC_STRIP, 0, 1, 0, 6 );
    CLT_VerTEXrGbA( &listPtr, 20.0, 20.0, 0.0, 0.0, 0.0, 1.0 );
    CLT_VerTEXrGbA( &listPtr, 23.0, 60.0, 0.1, 0.1, 0.1, 1.0 );
    CLT_VerTEXrGbA( &listPtr, 110.0, 10.0, 0.7, 0.7, 0.7, 1.0 );
    CLT_VerTEXrGbA( &listPtr, 95.0, 80.0, 0.7, 0.7, 0.7, 1.0 );
    CLT_VerTEXrGbA( &listPtr, 200.0, 20.0, 0.2, 0.2, 0.2, 1.0 );
    CLT_VerTEXrGbA( &listPtr, 190.0, 60.0, 0.14, 0.14, 0.14, 1.0 );
    ...
}
```

This piece of code outputs a strip comprised of six untextured, shaded triangles. The triangles start at roughly 20 pixels into the bitmap from the left, and move right until they are about 220 pixels in from the left edge of the bitmap. As they near the center, they get brighter.

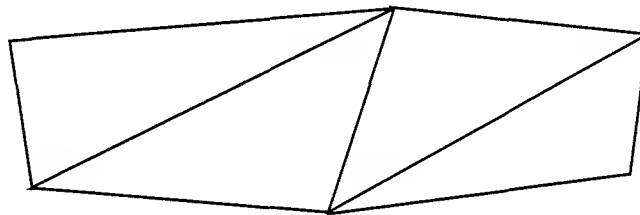


Figure 1-4 Triangle strip

State Control Instructions

The way in which a triangle is actually rendered in a bitmap is controlled by the state of the Triangle Engine. For example, the triangles in the previous sample code might have a texture blended with the shading, might be dithered, or might be rendered as partially translucent triangles. In this section, the methods for changing this state are described. The individual state-controlling registers are described in Chapters 3 and 4.

The state registers for the Triangle Engine are all 32-bits, although some features may only use some of the bits in a register, while others may be affected by data from more than one register. The Triangle Engine provides three ways of writing data to these registers, and so the CLT provides three methods as well:

```
CLT_WriteRegister( pp, reg, val )
CLT_SetRegister( pp, reg, val )
CLT_ClearRegister( pp, reg, val )
```

The first command, `CLT_WriteRegister`, stores the value `val` in the register `reg`. The second command, `CLT_SetRegister`, ORs the bits in `val` into the previous value of the register `reg`. The last command, `CLT_ClearRegister`, clears any bits in `reg` if the corresponding bits in `val` are set. For each of these calls, `pp` is the address of a pointer to the insertion point in a command list buffer.

These macros actually write a header word to the command list buffer, then the 32-bit quantity, `val`. This header word describes whether a Set, Clear, or Write should occur, at what address the change should occur, and how many consecutive registers are written. Each of the macros (`CLT_WriteRegister`, `CLT_SetRegister`, and `CLT_ClearRegister`) always write only one register at a time. If a user program wants to change the state of several registers at once, and these registers resided in consecutive addresses, a more efficient method would be to use:

```
CLT_WriteRegistersHeader( reg, cnt )
CLT_SetRegistersHeader( reg, cnt )
CLT_ClearRegistersHeader( reg, cnt )
```

Note that these CLT macros do not take `pp` as an argument, so user code has to look something like the example below:

```
{
    ...
    CLT_WriteRegister( pp, DBCONSTIN, 0xFFA01010 ); /* mostly red */
    *pp++ = CLT_WriteRegistersHeader( DBDITHERMATRIXA, 2 );
    *pp++ = 0xC0D12E3F; /* DBDITHERMATRIXA was just set */
    *pp++ = 0xE1D0302F; /* DBDITHERMATRIXB was just set */
    ...
}
```

Whenever these macros need to name a register, they can use one of the named register constants. In the CLT header file, all of the named registers are of the form `RA_<regName>`. For example, `RA_DBDITHERMATRIXA` is the register address of the top half of the destination blender's dither matrix.

Because some attributes might only use a few bits of one of the registers, the CLT header file also defines several bit shifts and masks that can be used to set just portions of a register. Each field within each register has been named in the form `FV_<regName>_<Attribute>`. For each of these field values, the shift and mask can be used to ensure that values do not write into other register attributes accidentally.

The following macros can be used in combination with the CLT_ClearRegister and CLT_SetRegister macros to only change a portion of register, so that the remaining fields within the register can remain intact.

```
CLT_Mask( reg, field )
CLT_Bits( reg, field, data )
```

The following code fragment sets only the red portion of the Destination Blender's BMultConstSSB0 register, to the value 0x40:

```
{
    ...
    CLT_ClearRegister( pp, DBBMULTCONSTSSB0,
    CLT_Mask( DBBMULTCONSTSSB0, RED ) ); /* Clear red channel */
    CLT_SetRegister( pp, DBBMULTCONSTSSB0,
    CLT_Bits( DBBMULTCONSTSSB0, RED, 0x40 ) ); /* OR in 0x40 for red */
    ...
}
```

For some of the field values, a constant makes more sense than a numeric value. These constants have also been defined in the CLT's header file, along with some macros to use them. The CLT_Const macro can be used to retrieve one of these constants to provide a value for the data field of the CLT_Bits macro. The following example illustrates one way to use the CLT_Const macro. It sets the InterFilter of the Texture Mapper to TriLinear filtering.

```
{
    ...
    CLT_ClearRegister( pp, TXTADDRCNTL,
    CLT_Mask( TXTADDRCNTL, INTERFILTER ) );
    CLT_SetRegister( pp, TXTADDRCNTL,
    CLT_Bits( TXTADDRCNTL, INTERFILTER,
    CLT_Const( TXTADDRCNTL, INTERFILTER, TRILINEAR ) ) );
    ...
}
```

For each of the state registers in the Texture Mapper and Destination Blender, the CLT header file also provides macros that take all of the necessary field values as arguments, and outputs the correct CLT_WriteRegister command. Using these simpler macros, in cases where all of the data in a register needs to be set at once, can make the code appear cleaner.

Flow Control Instructions

The Triangle Engine has some additional instructions that affect its data flow. These instructions, collectively known as the *flow control instructions*, start and stop the Triangle Engine, cause it to jump to a new place within a command list buffer, and can instruct it to actually load a PIP or texture from main RAM. The flow control instructions can use the same macros as the state control registers (described above) or any of the following macros:

```
CLT_Sync( pp )
CLT_Pause( pp )
CLT_Interrupt( pp, ival )
CLT_JumpRelative( pp, address )
CLT_JumpAbsolute( pp, address )
CLT_TxLoad( pp )
```

Caution: *It is important to understand the use of these registers, because using them incorrectly can cause the Triangle Engine to crash.*

The SYNC instruction tells the Triangle Engine to flush all of its internal pipelines. This command **must** be used to separate one or more consecutive triangle strips and/or triangle fans from any state control instructions that may follow the vertices. Failure to use the SYNC instruction, after vertices have been rendered, can lead to Triangle Engine crashes.

The PAUSE instruction works like the SYNC instruction, except that it also causes the Triangle Engine to stop at the word after the PAUSE. The TE remains stopped until it is restarted at some later time by the CPU. When a PAUSE is encountered, an interrupt can be generated if the Triangle Engine was set to cause an interrupt when it reached the end of a list. The command to tell the Triangle Engine to begin, once a PAUSE has been encountered, is a command that must be executed by the Triangle Engine Device Driver. Each command list buffer normally contains a PAUSE instruction after the last state control or vertex instruction in the buffer.

The INT (interrupt) instruction tells the Triangle Engine to cause an interrupt for the CPU. This should not be used by applications, since the Triangle Engine device driver currently does not handle special interrupts.

The JR (jump relative) instruction causes the Triangle Engine to jump to a different address within a command list to fetch instructions. The relative offset from the jump instruction (a 2's complement number), should first be loaded into the RA_DCNTLDATA register by using the CLT_WriteRegister macro. Note that this is normally not needed by applications.

The JA (jump absolute) instruction causes the Triangle Engine to jump to an absolute address to fetch further instructions. The address should first be loaded into the RA_DCNTLDATA register by using the CLT_WriteRegister macro. Note that this is normally not needed by applications.

The TLD (texture load) instruction is used throughout a command list buffer whenever it becomes necessary to load a new texture into the Triangle Engine's Texture RAM (TRAM). To load a texture successfully, it is normally necessary to set up several registers in the Texture Mapper first, such as the load address, etc. Once all of these registers are set up, you issue a TLD command before any vertex instructions attempt to use the new texture. Note that it is not necessary to SYNC immediately before or after a TLD instruction, unless it happens to be an instruction that follows a vertex instruction.

Command List Snippets

So far, this document has described the CLT macros as tools for writing into a command list buffer. There are many cases, however, where it is useful to build a block of Triangle Engine commands once, then re-use the pre-built data several times. One way to prevent your code from having to re-compute these macros, each time a repeated set of commands needs to be applied, is to write the commands to a *Command List Snippet*, or `CltSnippet` data structure. These snippets can be dynamically allocated and filled once with commands. Then, only a simple `memcpy` is necessary to place the data into a command list buffer. In the M2 Graphics Pipeline code, several command list snippets are saved off and re-used, to help save computing cycles. For example, one command list snippet is provided that disables texturing. Whenever a non-textured object follows a textured object, one simple call copies data into the command list buffer to disable the Texture Mapper. A `CltSnippet` is defined as follows:

```
typedef struct {
    uint32* data;
    uint16  size;          /* Number of words used */
    uint16  allocated;     /* Number of words allocated */
} CltSnippet;
```

To allocate a new command list snippet, call:

```
int CLT_AllocSnippet(CltSnippet *s, uint32 numWords)
```

This routine allocates enough memory for the specified number of commands to be placed into this `CltSnippet`.

There may be cases when it is useful to create one `CltSnippet`, then re-use variations of it several times. To copy one snippet into another, call:

```
void CLT_CopySnippet(CltSnippet *dest, CltSnippet *src)
```

The `dest CltSnippet` must already have been allocated, and its data area must be at least as large as the `src CltSnippet`.

Once these snippets are built, the routine that copies a command list from the `CltSnippet` into a command list buffer is:

```
void CLT_CopySnippetData(uint32 **dest, CltSnippet *src)
```

No check is done to ensure that there is enough space remaining in the command list buffer for the whole snippet, so it is the responsibility of the application to do this before attempting to copy the data into the command list buffer.

Once a command list snippet is no longer needed, the memory allocated to it can be freed with a call to:

```
void CLT_FreeSnippet(CltSnippet *s)
```

Graphics State (GState)

The Graphics State (GState) object helps users manage the command list buffers that store the data used by the Triangle Engine to render graphics.

The main job of the GState is to provide an easy way of creating command list buffers, filling them with Triangle Engine instructions, and eventually sending them to the Triangle Engine device driver, so that the instructions can be executed. The GState is designed to give the user flexibility in how memory is used for command list buffers while keeping the overhead, both in memory and in CPU cycles, to a minimum.

This chapter describes how to use the GState to create command list buffers, and how to correctly use them so that user code, the 2D and 3D Pipelines and Frameworks, and the Font Folio, can all work together to render graphics through the Triangle Engine.

This chapter contains the following topics:

Topic	Page Number
Introduction	12
Creating and Initializing a GState	12
Using a GState	13
Synchronizing the Triangle Engine to Video Signals through GState	14
Cleanup Routines	15
Combining CLT, the 2D and 3D Pipelines and Frameworks, and the Font Folio	15

Introduction

The 2D Pipeline and Framework, the 3D Pipeline and Framework, the Command List Toolkit, and the Font Folio, all write commands to the command list buffers. Providing one standard way for all of these components to write to and send command list buffers results in the following benefits:

- ◆ Commands from all of the different sources can be intermingled throughout the process of rendering a scene
- ◆ Memory used for command list buffers can be shared
- ◆ Code used to manage command list buffers can be shared

The GState is a data structure and collection of routines that encapsulate command list buffers in a way that all of these libraries and folios can understand and use. Routines are provided to create command list buffers, determine where the current insertion point is in a command list buffer, and to send a command list buffer to the Triangle Engine device driver. For optimal performance, command lists should be double-buffered, so that while the Triangle Engine is executing the instructions in one buffer, the CPU can begin filling the second buffer with the next batch of commands. The GState supports double-buffered command lists.

A GState object looks like the following structure:

```
typedef struct GState {  
    Err      (*gs_SendList)(struct GState*);  
    CmdListP gs_EndList;  
    CmdListP gs_ListPtr;  
} GState;
```

In addition, there are several private fields which must be accessed through the interface routines provided.

Creating and Initializing a GState

Before command lists can be created, a GState needs to be created. To create a GState, an application should call:

```
GState* GS_Create(void);
```

Memory is allocated for a GState structure, and the structure is initialized to a default state. Users should *not* allocate a GState as a stack variable, because the structure contains many fields that are private. The next step is to allocate one or more command lists on a GState. The following routine allocates the specified number of lists, and assigns them to a GState:

```
Err GS_AllocLists(GState* g, uint32 numLists, uint32 listSize);
```

Note that `listSize` is specified in WORDS, not bytes. Command lists are just blocks of user memory, like any other blocks allocated with `AllocMem()` or `malloc()`.

The other setup necessary for a GState is to specify what bitmap the command lists should be rendered into, and optionally, what bitmap to use as a Z-buffer. These buffers are set and retrieved with the following calls:


```

Err GS_SetDestBuffer(GState* gs, Bitmap* bm);
Err GS_SetZBuffer(GState* gs, Bitmap* bm);
Bitmap* GS_GetDestBuffer(GState* gs);
Bitmap* GS_GetZBuffer(GState* gs);

```

When double-buffering frame buffer bitmaps, it is usually desirable to create *tear-free rendering*. To accomplish this, an application can associate a signal bit with a GState and the Graphics Folio. When the Graphics Folio finishes displaying a bitmap, it sends a signal to a task, stating that the previously-displayed bitmap is completely off the screen, and thus safe to render into. If a signal is allocated for use as a display signal in the Graphics Folio, it can be associated with a GState by calling:

```
Err GS_SetVidSignal(GState* gs);
```

If the signal is set on a GState, the GState waits before sending the first command list buffer for a bitmap.

Using a GState

Once a GState has been set up, data can be written to its command list buffers by writing 32-bit words to a pointer to the current insertion point in the current command list buffer. The GState maintains such a pointer in the field `gs_ListPtr`. Most of the Command List Toolkit macros, that write data to a command list buffer, take the address of this field as one parameter, and update the `gs_ListPtr` as they write data to the buffer.

The 2D and 3D Pipelines already do their own checks to ensure that there is enough space in a command list buffer before issuing a block of commands. However, when using CLT, it is up to the application to ensure that there is enough room remaining in the buffer. The following call ensures that there is enough memory for the requested number of words of space:

```
void GS_Reserve(GState* gs, uint32 numWords);
```

If sufficient space isn't available, the current command list buffer is sent to the Triangle Engine, and the next available command list is made current. The `gs_ListPtr` field then points to the beginning of this new command list buffer. Note that `GS_Reserve()` will fail if an application requests more words than are actually available in an entire list. If this happens, the results are unpredictable. It is a good practice, while developing an application, to allocate plenty of space to the command list buffers. You can reduce the size of the buffers once the rest of the code has stabilized. At this point, you can make a good estimate of what the minimum buffer size needs to be. Note that `GS_Reserve()` does not actually advance the insertion point by any amount of space. It is up to user code, or CLT macros, to advance the insertion pointer as data is written to the buffer.

When `GS_Reserve()` does not find enough space available in the current command list buffer, it calls the function pointed to by the `gs_SendList` field. This field normally points to the following routine:

```
Err GS_SendList(GState* gs);
```

This routine can be called at any time by the application. It flushes the current command list buffer to the Triangle Engine. `GS_SendList()` sends a command list buffer to the Triangle Engine to be rendered. This routine can optionally wait for a video signal saying that it is safe to start rendering to a buffer, and will ensure that `gs_ListPtr` is pointing to an available command list buffer before

returning. `GS_SendList()` calls `SendIO` to send the command list buffer to the Triangle Engine device driver. The following routine can be used to wait for all current and pending IORequests to complete:

```
Err GS_WaitIO(GState* gs);
```

Normally, this routine is called before attempting to display a buffer in a double-buffering scheme, so that the application can ensure that Triangle Engine rendering is not visible to the end user.

If it becomes necessary to write your own routine to send a list to the Triangle Engine, the following few routines might be helpful:

```
CmdListP GS_GetCurListStart(GState* g);
```

This routine returns a pointer to the beginning of the current command list buffer. In order to send a list to the Triangle Engine device driver, it is necessary to specify the start of a command list buffer, the end of the buffer, and some frame buffer information. Look at the source for `GS_SendList()` for exact details on how to send a command list.

```
void GS_SetList(GState* g, uint32 idx);
```

`GS_SetList()` sets which command list to use for the given `GState`. The `idx` field is in the range of 0...(num cmd lists - 1).

```
uint32 GS_GetCmdListIndex(GState* g);
```

This routine returns the index of the command list currently in use. Normally, you would call `GS_SetList()` with `(this index + 1) % (num cmd lists)` after sending the current list.

Synchronizing the Triangle Engine to Video Signals through GState

As mentioned earlier, it is possible to set up a `GState` to provide tear-free double-buffering of frame buffers. The basic principle is: Before a bitmap can be used as an output buffer, the `GState` ensures that it is not currently being displayed on the screen. The Graphics Folio can send signals that, if used correctly, allow the `GState` to always write only to buffers that are not up on the display. In combination with the correct calls to the Display Folio, buffers are swapped onto the screen with no visible tearing. The basic steps to follow to achieve this tear-free double-buffering are:

- ◆ Allocate a signal bit by calling `AllocSignal()`.
- ◆ When a view is created, define this signal as the display signal. This is done by specifying the signal as the argument for the `VIEWTAG_DISPLAYSIGNAL` tag when calling `CreateItem` for the view.
- ◆ Create a `GState` by calling `GS_Create()`.
- ◆ Associate the signal with the `GState`. This is done by calling the routine:

```
Err GS_SetVidSignal(GState* gs, int32 signal);
```
- ◆ At the beginning of each frame to be displayed as a frame buffer, call:

```
Err GS_BeginFrame(GState* gs);
```

If rendering to a buffer that is not going to be immediately displayed, do not call `GS_BeginFrame()`. When `GS_SendList()` sees that `GS_BeginFrame()` has been called, and a signal is associated with a `GState`, then `GS_SendList()` waits

until that signal is activated by a call to `ModifyGraphicsItem()`. If a bitmap is going to be rendered into, and `ModifyGraphicsItem()` is not called to display it, the signal is never sent to the GState, and the application effectively hangs.

Cleanup Routines

To free the memory allocated to the command list buffers for a GState, call:

```
Err GS_FreeBuffers(GState* gs);
```

If, for some reason, it is necessary to re-allocate new command list buffers, `GS_AllocLists()` can be called again.

Once an application is completely finished using a GState, the GState can be freed by calling:

```
Err GS_Delete(GState* gs);
```

If `GS_FreeBuffers()` has not been called prior to `GS_Delete()`, it is called implicitly as the GState is freed in memory.

Combining CLT, the 2D and 3D Pipelines and Frameworks, and the Font Folio

The 2D and 3D Graphics Pipelines, and the Font Folio, are all built on top of GState. One benefit of this is that calls from these various sources can be intermingled throughout a frame. For example, if an application is using the 3D Pipeline to render some scenery for a flight simulator, and then going to use the Font Folio to display a status bar with altitude, air speed, etc., the application could set up the scene, and tell the 3D Pipeline to render it. Before the command lists are flushed, the application could then call the Font Folio routines to display the appropriate text. A final call to `GS_SendList()` would then render all of the commands from the Pipeline and the Font Folio into the bitmap at once.

Textures

In M2, a *texture* is a two-dimensional image that is used to modify the color or the transparency of a 3D object. A texture follows the surface that it is mapped to—often folding, bending, or wrapping around to follow the contours of a 3D object.

This chapter contains the following topics:

Topic	Page Number
Introduction	18
Texture Mapping	19
Mipmap Filtering	21
Using Mipmap Filtering	24
Perspective Correction	27
How Textures Are Stored in Memory	28
Texture Formats	29
How Texels Are Stored in Memory	29
PIP Tables	32
Texture Application Blending	34
Texture Mapping Step-by-Step	35
Loading Textures	36
Texture Mapper Pipeline Register Definitions	40

Introduction

A texture can be something as simple as a color (with or without various degrees of shading) or something as complex as (or more complex than) a tiled pattern, a newspaper page, a brick wall, or even a previously-rendered frame buffer. The texture pipeline is capable of processing from 1 pixel every 3 ticks to 2 pixels per tick, depending on the filtering being applied.

In the M2 system, textures are stored in arrays, and each element of the array in which a texture is stored is called a *texel*. Each texel can be up to 32 bits long. For more information about the structure of a texel, see "How Textures Are Stored in Memory."

The coordinate system for a texture is shown in Figure 3-1. By convention, the horizontal and vertical coordinates of a texture are expressed as U and V , as shown in the diagram. If U is a variable that is used to represent horizontal coordinates and V is a variable that is used to represent vertical coordinates, the point where $\langle U, V \rangle = \langle 0, 0 \rangle$ is at the top-left hand corner of the texture's mipmap, and the address of that point is considered the base address of the texture. This means that in the case of an 8-bit texture, the map is stored in memory with U increasing by one from one memory location to the next.

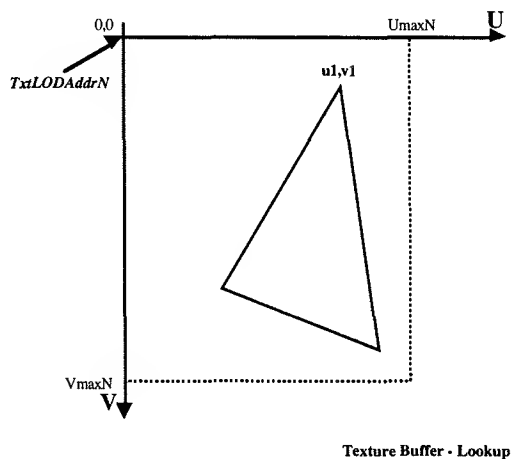


Figure 3-1 *Texture Coordinates*

Note: U and V coordinates must be in the range from 0 to 1023.

For more information about how M2 stores textures in memory, see "How Textures Are Stored in Memory." See Appendix A for the register setup used to turn texturing on and off.

Texture Mapping

Texture mapping is the process of mapping a texture onto a 3D object displayed on a screen. Texture mapping can be tricky because a texture is a 2D rectangular region that must often be mapped to a 3D non-planar surface with various kinds of irregular features. For example, Figure 3-2 shows how a texture might be mapped to a pyramid-shaped 3D object.

The texture shown in the rectangle on the left has been mapped to the pyramid shown on the right. Notice that the texture obtained from the texture on left is wrapped around to match the shape of the pyramid and is distorted when it is mapped to the pyramid.

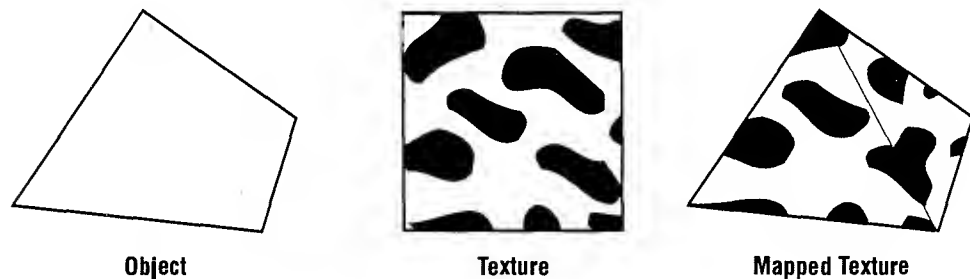


Figure 3-2 The texture-mapping process

Using Filtering in Texture-Mapping Operations

Depending on several factors—including the size of the texture being used, the 3D object's distortion, and the size of the screen image—some texels in a texture might be mapped to more than one pixel in the object's image, and some parts of the object might be covered by multiple texels.

Because a texture is made up of discrete texels, the M2 Triangle Engine performs various kinds of filtering operations to map texels to pixels in images of 3D objects. For example, if many texels correspond to a single fragment of an object, they are averaged down to fit the number of pixels into which they are mapped. If texel boundaries fall across fragment boundaries, things get even more complicated, and a weight average of the affected pixels is performed.

Replicating Textures

M2 allows you to repeat a texture over the surface of an object. Replicating a texture in this way is called *tiling*. If a texture is to be tiled, it must have a size that is a power of two in both directions. If this condition is met, a simple mask is used on the texture's final *U*, *V* coordinate (refer to Figure 3-1) before the address calculation is performed to mask out some most significant bits (MSBs).

Mipmapping

A textured object, like any other object, can be viewed at different distances from the camera. If a textured object moves away from the viewer, or if the viewer moves away from the object, the number of pixels covered by a single texel decreases. To make this change in texture mapping correlate properly with the reduction of the object, M2 filters the texture map down to an appropriate size as the object grows smaller, without introducing any visually disturbing artifacts.

To prevent the generation of such artifacts, M2 uses prefiltered texture maps called *mipmaps* (see Figure 3-3).

Mipmaps (MIP stands for the Latin *multum in parvo*, meaning “many things in a small place”) are maps that provide multiple sizes of the same texture in powers of 2. The size of a mipmap can range from 1 x 1 texel to the size of your highest-resolution map. For example, a set of four mipmaps could have sizes of 64 x 16, 16 x 4, 8 x 2, and 4 x 1.

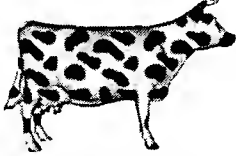
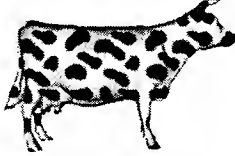






Mipmap Level	Resolution	Actual Size	Magnified
0	Full		 <i>Not Magnified</i>
1	1/2		 <i>Magnified by 2</i>
2	1/4		 <i>Magnified by 4</i>
3	1/8		 <i>Magnified by 8</i>

Figure 3-3 *Mipmaps*

Figure 3-3 shows how mipmaps provide multiple sizes of the same texture image. Notice that each mipmap is one-fourth the size of the next higher-level mipmap: one-half its height, and one-half its width.

Levels of Detail (LODs)

Each version of a texture's image is called is called a *level of detail* (LOD). The original version of the texture, which has the finest detail, is called LOD 0. Other LODs are numbered upward consecutively, as shown in Figure 3-3.

M2 hardware supports the use of up to four levels of detail (LODs) at a time, but you can control which LODs are used by specifying the index of the finest LOD and the number of LODs to use.

Using Mipmaps

When you have created a set of mipmaps, here's how M2 uses them: Starting with the highest-resolution texture, the desired number of filtered and minified versions is created. Each version is reduced from the size of the previous version by a factor of two in both the *U* and *V* directions (*U* and *V* are the texture-coordinate equivalents of *x* and *y* in the world coordinate system).

When an object is texture-mapped and a set of mipmaps is available, M2 can automatically determine which mipmaps to use, on a per-pixel basis, based on the size (in pixels) of the object being mapped. When the image of the object gets small enough, M2 can switch to the next smallest mipmap to map the object's texture. When the object grows large enough, M2 can switch to the next-largest mipmap.

Mipmap Filtering

Mipmapping improves the appearance of rendered images considerably. But when you map a texel to a surface, there is rarely an exact match between the (*U*, *V*) coordinates of the texel being mapped and the (*x*, *y*) coordinates of the mipmap being used in the mapping operation. In other words, the precise texture-to-map ratios shown in Figure 3-3 are ideal ratios that actually seldom occur.

When that is the case—as it usually is—an operation known as *mipmap filtering* is often required to avoid aliasing.

Although some mipmap filter is needed in most texture-mapping operations, the amount of filtering you should perform in any particular situation can vary, depending on your needs. Also, M2 supports four different filtering modes, each designed for use in a specific kind of situation.

The four filtering modes offered by M2 are:

- ◆ *Nearest (point) filtering*, in which each texel in a texture is mapped to a surface using the best mipmap that is available, ignoring the possibility that the match may not be precise.
- ◆ *Linear filtering*, which uses two mipmaps—one that is actually too large and another that is actually too small—to average the color of each texel being matched
- ◆ *Bi-linear filtering*, which uses a similar technique to the one used in linear filtering, but applies a weighted average of the colors of the texels surrounding the texel being mapped in order to obtain a color value for that pixel that is more precise
- ◆ *Quasi tri-linear filtering*, which uses a slower but more precise algorithm to obtain an even more precise weighted value for the texel being mapped.

Each of these filtering modes has advantages and disadvantages. As you move from the fastest filtering mode (nearest [point] filtering) to the slowest, the quality of the rendered image improves, but the time required to filter the data increases. Nearest (point) filtering is the fastest mode, but offers the lowest resolution. Quasi tri-linear filtering is the slowest method, but offers the highest resolution.

Table 3-1 lists the four texture-filtering modes used in M2 and compares their speed and characteristics. Note that the bi-linear and quasi tri-linear filtering modes require a 2-by-2 array of texels each.

Table 3-1 Texture Filtering Modes

Type	Speed	Texels Required	Operation
Point	2 pix/tick	$T[U, V, n]$	None $\text{Out} = T[U, V, n]$
Linear	1 pix/tick	$T[U, V, n]$, $T[U/2, V/2, n+1]$	When a sample lies between two mipmaps, the distance of the sample from each map is used to compute a linear blend between the two maps.
Bi-Linear	0.5 pix/tick	$T[U, V, n]$, $T[U+1, V, n]$, $T[U, V+1, n]$, $T[U+1, V+1, n]$	The fractional placement within a map is used to blend four adjacent texels from one map together.
Quasi Tri-Linear	0.33 pix/tick	$T[U, V, n]$, $T[U+1, V, n]$, $T[U, V+1, n]$, $T[U+1, V+1, n]$, $T[U/2, V/2, n+1]$	A mixture between the two above modes. A bi-linear blend is performed at LOD_n , then a linear blend is performed with a texel from LOD_{n+1} .

Note: For texels at the edge of a texture, all the information required may not be present. You may be able to ignore this anomaly, or you may want to add a one-pixel guard region around the whole texture. By adding a guard region, you can guarantee that all required information is present. (A guard region is actually required for texture tiling; for details, see "Runtime Carving" on page 38.) In the M2 hardware, the address clamps in both the U and V directions, so if you do not add a guard region, replicating the last texel is the default mode.

How M2 Mipmap Filtering Works

Figure 3-4 shows the various kinds of filtering operations that are used for mipmapping in M2. The diagram is a stylized representation of a perspective view that extends from the camera out into the distance. Such a triangle could exist, say, as part of the floor of a large building.

The problem that such a triangle generates for the M2 hardware is that at some points the camera is not between two LODs. At those points, linear interpolation between maps is not possible. (It is assumed here for sake of argument that Region 3 in the diagram is the desired operating region. At that point, the camera lies between maps of different LODs, so you can perform linear or tri-linear interpolation).

If the sampling process takes you into Region 2, that means you are trying to magnify beyond the finest level of detail, which is not possible. That is the point where interpolation no longer works, and where point sampling the texture would lead to blockiness. At that point, as in the previous illustration, the best you can do is to switch to bi-linear interpolation in such a way that the fraction bits of the (U, V) coordinates are used to position a box filter over four neighboring texels.

Note that this alternative does not create any more information; it simply blurs the information. But that is preferable to the blockiness that would result from point sampling.

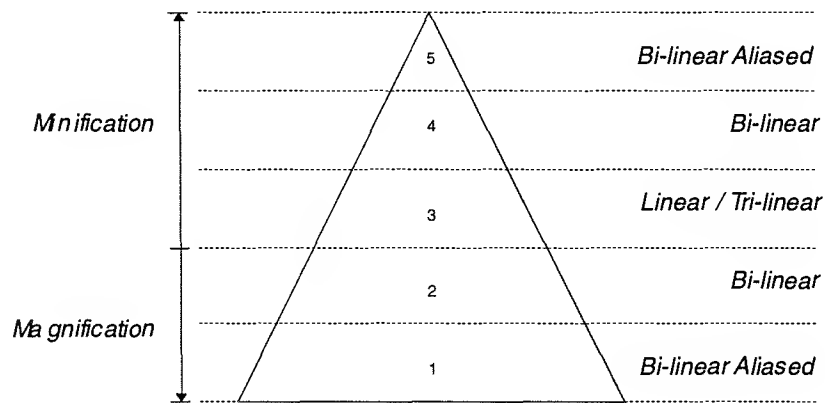


Figure 3-4 Tradeoffs in filtering operations

Monotonic Region Variations

In Figure 3-4, notice that within any one span, the region variation is *monotonic*—that is to say, once you are out of one region, no more pixels within the horizontal span you have entered can fall within that region again. It is possible, however, to jump past several regions at once. For example, you can go from Region 1 to Region 5 in one pixel step. However, you can't go from Region 3 to Region 4 and back to Region 3 again within a span.

Figure 3-4 also illustrates some problems that can arise in using mipmap filtering. Assume that the scene shown in the diagram has such a long perspective that the texture you are applying to the nearest rectangle is magnified beyond the level of detail provided by the finest mipmap available. At the same time, assume that the texture you are applying to the farthest rectangle is minified beyond the level of detail provided by the coarsest mipmap available.

Problems in Minification

For minification, a problem arises if the programmer has not provided all the LODs that are needed to bring resolution down to the 1-by-1 map. Again, the M2 hardware can detect the fact that it has fallen out of the range in which linear interpolation is possible. When that happens, the hardware falls into Region 4, where it must now either point-sample the coarsest available LOD map (an operation results in aliasing) or (once again) use bi-linear interpolation.

Because the hardware can easily detect any of the conditions that have just been described, it is possible for the hardware to select the best possible algorithm for each region. But blindly trusting the hardware to make such a selection has drawbacks that arise from the performance differences between the various interpolation schemes.

If you don't want to trust everything to the hardware, you can specify (by setting a register) exactly which algorithm to select for each region. Table 3-2 shows all the legal selections you can make. Note that regions 1 and 5 are really part of 2 and 4 respectively.

Table 3-2 Registers for specifying LOD Regions

Region	Point	Linear	Bi-Linear	Q Tri-Linear
1	•		*	
2	•		*	
3	•	•	*	•
4	•		*	
5	•		*	

Warning: For a filtering operation that involves perspectives such as those shown in Figure 3-4 on page 23, you will probably make the boundaries between regions visible if you change filter modes. When you are working with a static image, this side effect may not be very noticeable, but in a moving image the effect can be serious more serious because each region will have different motion artifacts.

Using Mipmap Filtering

Now that we have seen how mipmap filtering works, we are ready to examine each of the varieties of filtering available in M2: nearest (point) filtering, linear filtering, bi-linear filtering, and quasi tri-linear filtering. This section describes each of these kinds of filtering in more detail.

Nearest (Point) Filtering

Nearest (point) filtering is the simplest kind of filtering operation. You can always perform nearest (point) filtering.

In a nearest (point) filtering operation, the filtered texture values map directly to the color and alpha values of the texel containing the point. This point is represented by the coordinates (U, V) in the equation below.

$$LOD = [U, V, n]$$

U and V are the texel coordinates and n is the level of detail (LOD) of the mipmap being used for the mapping operation.

Linear Filtering

When a filtering operation is performed, the (U, V) coordinates for adjacent destination pixels may be such that the computed level of detail falls between two LODs. For example, assume that i is the distance of the incrementation between two horizontal texels being mapped to a surface. If $(U(i+1) - U(i))$ is 3, the computed level of detail is in between 1 and 2. In such a case, linear filtering can be used to blend the texel data between the two bounding LODs. Linear filtering is illustrated below.

$$T = T[U, V, n] \cdot (1 - \text{blend}) + T[U, V, n + 1] \cdot \text{blend}$$

n is the level of detail (LOD) of the mipmap being used for the mapping operation, just as it was in the previous equation.

Linear filtering works by taking samples of the texture at (U, V, n) and at $(U, V, n + 1)$. The sample values are combined in the ratio provided by the *blend* variable, which is computed based on the distance of the computed LOD from the finer LOD, in accordance with the preceding equation.

Requirements for Linear Filtering

Because linear filtering interpolates between two levels of detail, it cannot always be used for all pixels in a rendered triangle. In order to use linear filtering, the following requirements must be met:

- ◆ The texture being applied must have more than one LOD.
- ◆ The pixels being rendered must scale the texture such that it is larger than the finest LOD, and smaller than the coarsest LOD.

An Example of Linear Filtering

As an example of linear filtering, assume that a triangle is being rendered using a texture map that contains two levels of detail. Also assume that the dimensions of LOD 0 are 20×20 texels. That means that the dimensions of LOD 1 are 10×10 , because each subsequent LOD must be half the width and half the height of the previous LOD. So the triangle to be rendered always uses texture coordinates that range from (0,0) to (19,19).

Note: For the sake of simplicity, this example ignores the effects of the perspective correction factor, W , and assumes that an increase of 1.0 in triangle coordinate space maps directly to an increase of 1.0 in screen coordinates.

Now assume that this triangle is to be rendered in such a way that its screen dimensions will be larger than 20 pixels wide or 20 pixels high. If that is the case, the largest texture LOD will have to be scaled *up* to fit the triangle. In M2 terminology, we say that the *magnification filter* is used.

In contrast, if the triangle is rendered in such a way that its screen dimensions are smaller than 10 pixels wide or 10 pixels high, the smallest LOD must be scaled *down* to fit the triangle. In M2 terminology, we say that the *minification filter* is used.

Now let's consider a third case. If the triangle we are discussing is rendered in such a way that its screen dimensions are between 10 and 20 pixels in the horizontal and vertical directions, the M2 *interfilter* is used. Note that when the Triangle Engine determines which LOD is needed, horizontal LOD and vertical LOD are computed independently, so it is possible that a pixel gets magnified horizontally, and minified vertically.

There is only one case in which linear filtering can occur (or, to put it into M2 terminology, there only one case in which the *linear filter* is used). That case arises when there is a texture map larger than the rendered triangle's needs, or when there is a texture map smaller than the rendered triangle's needs. When either of those texture maps is needed, the M2 linear filter can interpolate between the two. When a texture map needs to be magnified and there is no larger texture map available to interpolate the finest LOD, or when a texture map needs to be minified and there is no smaller texture with which to interpolate the coarsest

LOD, the linear filter can be used. Because certain filter types only make sense when a texel is between LODs (in other words, when the interfilter is used), each of the three filter regions can independently select one of the available filter modes.

Note: Linear filtering takes twice as long as nearest (point) filtering.

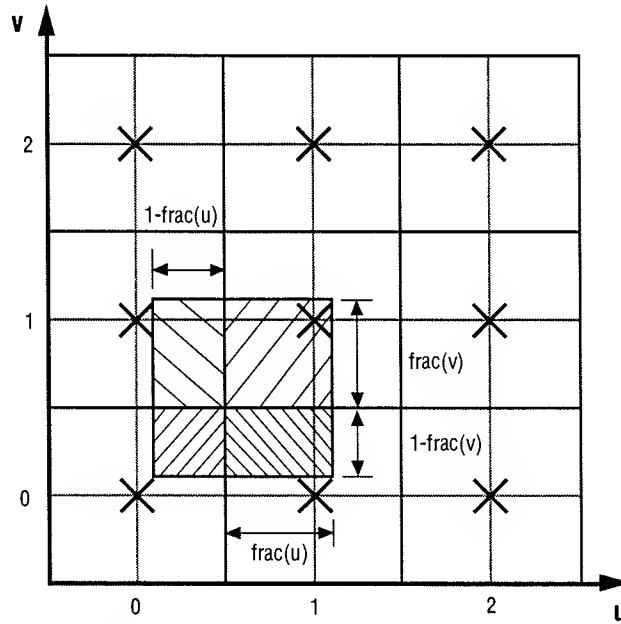


Figure 3-5 Bi-linear filtering.

Bi-linear Filtering

Bi-linear filtering (Figure 3-5), like nearest (point) filtering, is a type of filtering that can always be performed. Bi-linear filtering is equivalent to a one-pixel area filter that takes the weighted average from four adjacent texels, as shown below.

$$T_{out} = T[U, V, n] * (1 - fu) * (1 - fv) + T[U, V + 1, n] * (1 - fu) * fv + \\ T[U + 1, V, n] * fu * (1 - fv) + T[U + 1, V + 1, n] * fu * fv$$

In a bi-linear filtering operation, the fractional parts of (U, V) are used to position the center of the filter, as shown in Figure 3-5. Note that the texel center is at $(.0, .0)$. Given the fractional parts of U and V , the four shaded areas can be constructed. The output value is then the sum of all four fractional areas multiplied by the corresponding pixel values.

Note: Bi-linear filtering is four times slower than nearest (point) filtering. For a precise comparison of the speeds of different filtering modes, see Table 3-1.

Quasi Tri-linear Filtering

Quasi tri-linear filtering is a cross between linear filtering and bi-linear filtering. In a quasi tri-linear filtering operation, a bi-linear blend is performed at the closest level of detail (determined by looking at the blend value) and point sample a texel

at the further LOD. A linear blend is then performed on the two results. This procedure results in the highest-quality resampling of the texture, but is somewhat slower than a straight bi-linear blend (see Table 3-1).

Quasi tri-linear filtering provides the best possible texturing quality, but it cannot always be used for all pixels in a rendered triangle. Because quasi tri-linear filtering performs a linear blend between two LODs, it can only be specified as the interfilter.

Perspective Correction

The M2 Triangle Engine has two modes of operation. The default mode is called *Perspective On Mode*. In this mode, M2 calculates the texture locations with perspective correction. The second mode is called *Perspective Off Mode*.

Perspective On mode provides perspective-correct U and V coordinates that are appropriate for mapping a texture onto a 3D surface. In Perspective Off mode, the texture values that are passed to the Texture Mapper are the raw U/w and V/w values, before division by $1/w$. Perspective Off mode can be useful in a 2D environment when you want to use a texture as a sprite and map it directly to the screen.

See Appendix A for the register setup used to turn perspective correction on and off.

Computing the U and V Coordinates for Texture Mapping

In Perspective On mode, the U and V coordinates are typically computed as

$$(U/w) / (1/w)$$

and as

$$(V/w) / (1/w)$$

This computation results in perspective-correct U and V coordinates that are appropriate for mapping a texture onto a 3D surface. When the Perspective Off mode is selected, the $1/w$ value is not used, and the U/w and V/w coordinates are passed directly into U and V .

Using Perspective Off Mode in 2D Operations

Perspective Off mode can be useful in 2D rendering when the user wants to step through a flat texture and map it directly to the screen. It can also be useful in 3D mode at extremely large distances.

You can use Perspective Off mode when you want to generate successive integer U and V values for each succeeding pixel of a span.

The easiest way to accomplish this goal is to use Perspective Off mode, but to set the U/w and V/w values at the vertices of the triangles in such a way that the d/dx values of these parameters are the same value: namely, 1.

You can accomplish this same effect in Perspective On mode by setting the $1/w$ value as close to 1 as possible, making $(U/w) / (1/w)$ is basically equal to U/w . However, this technique precludes the use of the $1/w$ value for z-buffering in this situation. If z-buffering is not needed, using Perspective Off mode can eliminate the need for passing any $1/w$ information into the Triangle Engine at all.

How Textures Are Stored in Memory

The M2 hardware supports texture sizes of up to 1,024 by 1,024 texels and up to four levels of detail (LOD). All the LODs that you use simultaneously must fit into a 16k block of dedicated RAM within the Triangle Engine called texture RAM, or TRAM.

In M2, the TRAM must be demand-loaded by software. Each successively coarser LOD must be exactly half the size in *U* and *V* from the previous level.

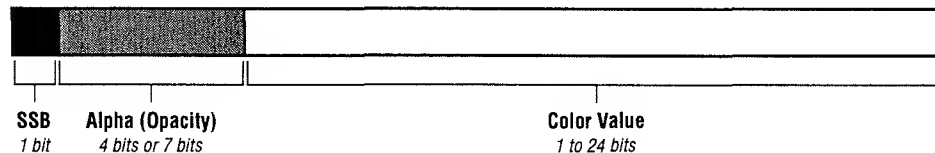
The programmer specifies the size of the coarsest LOD and the number of LODs to load. The size for the finer LODs are calculated automatically. Any aspect ratio texture may be specified.

When you load a texture into TRAM, you must load all adjacent LODs. For example, you cannot load just LOD0 and LOD2—you must also load LOD1. It is important to comply with this requirement; if you do not, the hardware produces unpredictable results.

When a texture is loaded, the *Umax* and *Vmax* register fields (see TXTUVMAX - Texture Loader Width Register (0x0004_642C)) are loaded with the dimensions of the coarsest map that is loaded. The finer mipmap sizes are given by multiplying by $2^{(LOD_{max} - LOD_N)}$ where *LODN* is the current level of detail. Each texel in a texture can contain from one to three components. If a texel has three components, they are:

- ◆ A *source selection bit* (SSB) that is generally used to select between two sets of constant registers (for more information, see "Texture Mapping"). An SSB, when used, is always 1 bit long.
- ◆ An *alpha component* that specifies the texture's opacity. An alpha component can be either 4 bit long or 7 bits long.
- ◆ A *color component* that has two possible uses. It can specify the colors used in the texture or can be treated as an index into a palette index table (see "PIP Tables"). A color component can be 1 to 24 bits long.

The length of a texel's alpha component and color component depend on two things: whether the texel is compressed or uncompressed, and what kind of information the color value of the texel contains. illustrates the three parts of a texel.



The three possible components of a texel

Texture Formats

Textures in memory may be of two types: compressed or uncompressed. Texels within a texture may be of two types: literal or indexed. The term *texture formats* encompasses all four of these texture types. Table 3-3 lists and describes all four texture formats used in M2.

Table 3-3 *Texel Types*

Type	Description
Literal	Texels are stored as real color values. Only two formats are supported: five bits per color component or eight bits per color component.
Indexed	Texels may be stored using an arbitrary number of bits per texel up to eight bits. These get expanded to 8 bits per color component by using the PIP.
Compressed	Texels are stored using a run length encoding. Multiple bits per texel (types) may be used within the texture. These are expanded to the largest texel size used when the texture is read into the internal memory. Literal formats can not be used in compressed textures.
Uncompressed	Texels are stored in one format only. The number of texels is exactly the height * width.

In addition to color information, each texel format shown in Table 3-3 may also have two more pieces of information associated with it: alpha (which may be four or seven bits), and a Source Select Bit (SSB), which is a one-bit value that has multiple uses (in general, the SSB is used to select between two sets of constant registers for each stage defined for each register; for more details, see "How PIP Tables Work."

Table 3-4 lists all the combinations of color, alpha and SSB values that a texel can have.

Table 3-4 *Color, Alpha, and SSB Values that Can Be Assigned to a Texel*

Color	0	1	1	2	0	3	4	6	5	1	2	0	3	4	7	8	4	7	8	8	15	24	24
Alpha	-	-	-	-	4	-	-	-	-	4	4	7	4	4	-	-	7	4	4	7	-	-	7
SSB	1	-	1	-	-	1	-	-	1	1	-	1	1	-	1	-	1	1	-	1	1	-	1
Total	1	1	2	2	4	4	4	6	6	6	6	8	8	8	8	8	12	12	12	16	16	24	32

How Texels Are Stored in Memory

Uncompressed texels and compressed texels are stored in memory in different ways. Uncompressed texels are stored as arrays of texel data, but compressed texels are stored as texels of different types that are run length encoded. This section describes both these methods of storing texels in memory.

How Uncompressed Texels are Stored

Uncompressed textures are stored in memory as arrays of texel data. The number of bits per texel may vary from 1 to 32. Figure 3-6 shows how uncompressed texels are stored in memory.

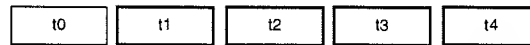


Figure 3-6 Storage format of an uncompressed texel

How Compressed Texels are Stored

Compressed textures consist of texels of different types (or formats) of texels that are run-length-encoded. This storage format allows the source texture to contain texels of different depths, so that portions of a texture that need more detail can use higher texel depths.

When texels are stored in memory, each sequence of texels that have the same type and value is called a *run*. Each new run is preceded by a control byte that specifies the run length and the type. Portions can be encoded to use less memory by switching to a different texel format on a per-run basis.

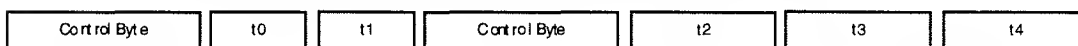


Figure 3-7 Storage format of a compressed texel

Figure 3-7 shows two run lengths of different texel types. As you can see, each run length is preceded by a control byte (in this example, the different run lengths have different depths as well).

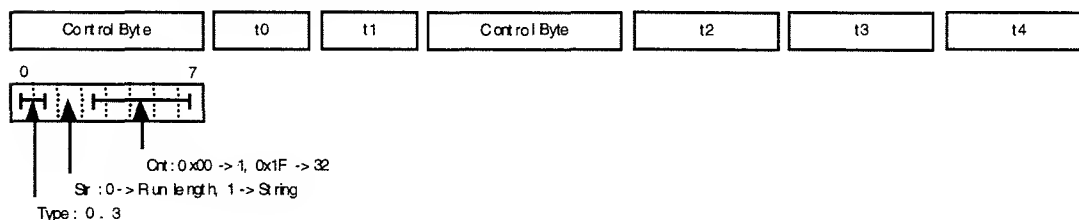


Figure 3-8 Bits in the control byte

Figure 3-8 shows the bits in the control byte that precedes each run length in a compressed texel. As the diagram shows, a control byte has three fields. These three fields are defined as follows:

- ◆ *Type (bits 0 and 1)*—This field specifies the texel type (format) of the run of texels that immediately follow the control byte. The TYPE field is simply used to select which of the *TxtLdSrcForm* registers to choose. Four types are possible. Each type is specified by the values of the four *TxtLdSrcForm* registers. The next control byte may specify a different type for its run. All types are expanded out to the format specified within the *TxtExpForm* registers.
- ◆ *Str (bit 2)*—This bit specifies two kinds of storage: *string* storage when set, and *run-length* storage when cleared. When string storage is used, the CNT field (next item in this list) specifies the number of texels that follow the control

byte. When a run is copied into TRAM, each of these texels is copied to the TRAM—that is, n texels in the source are mapped to n texels in the TRAM). When run-length storage is used, the control byte is followed by only one texel. This texel is copied into the next n locations within the TRAM, where n is encoded in the CNT field (i.e. one texel in the source gets expanded to n texels in the TRAM).

- ◆ *Cnt (bits 3 through 7)*—This field encodes the number of texels that the current control byte will generate. If n is the total number of expanded texels, then $n = \text{CNT} + 1$. If STR is set to *string*, n also specifies the number of texels following the current control byte.

Loading a compressed texture requires two steps: Run-length decoding and texel decompression. These steps work as follows:

1. *Run-length decoding* searches for control bytes and extracts the appropriate number of texels per control byte.
2. *Texel decompression* is the process that converts the four possible source texel formats to one format that is placed in the TRAM (the TRAM may only contain one texel type).

When you store compressed texels in memory, you can use source texels that are chosen from any of the legal texel types (indexed and literal) and may be mixed to some extent within the source texture (specifically, up to four different formats can be mixed and matched per texture). The restrictions on the use is that literal and indexed texels may not be mixed within one texture.

Special Settings in the Control Byte

Two special settings can appear in the control byte shown in Figure 3-8:

- ◆ If the setting of the *Str* field is 0 and the setting of the *Cnt* field is 0, 0, the run that follows the control bit has a run length of 1. This setting is reserved, and should not be used in applications.
- ◆ If the *Type* field is programmed as transparent—that is, if $\text{TxtLdSrcForm}[\text{TYPE}].\text{Trans} = 1$ —the *Str* and the *Cnt* fields are concatenated to form one 6-bit count value. No source texels follow the control byte. When a run uses a transparent format, the source texels are retrieved from a corresponding user-settable constant. See the "Texture Mapper Pipeline Register Definitions" section, later in this chapter, for more information.

If a texture in main memory is run-length encoded, it can be a mixture of the preceding formats (although literal and indexed formats cannot be mixed). The texture within the TRAM may only contain one format at a time. It is the job of the loader to ensure that only one texel type is used in the TRAM. The filtering and blending stages all use a consistent 1, 8, 8, 8, 8 format for SSB, Alpha, Red, Green and Blue.

PIP Tables

A PIP table (palette index table) is a component of the M2 Triangle Engine that is used to look up information about textures that are stored as indices. From the information stored in a PIP table, you can reconstruct the stored texels at load time.

Every PIP table has 256 entries. Each PIP-table entry is a 32-bit value constructed just like a texel. When you use a PIP table to look up information about a texture, the input texel is used as an offset into the table that contains the desired palette for the texture. The entries in a PIP table do not have to be expanded out to full 32-bit colors before filtering or texture application can be performed.

Figure 3-9 shows how a texel's color component can be used as an index into a PIP table.

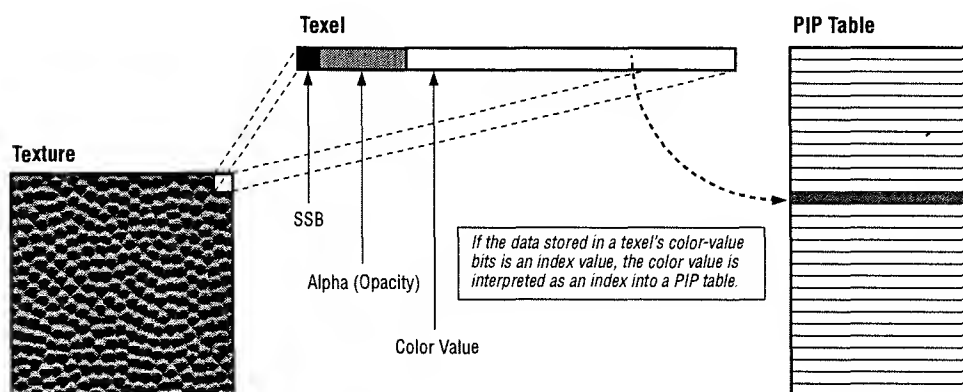


Figure 3-9 How textures and texels work with PIP tables

PIP tables have a fair amount of built-in flexibility. The format of input texels can range from 0 to 8 bits per texel. An alpha value may also be present. Adding an alpha value increases the number of available formats, but the alpha value is not used to reference into the PIP table. The way in which the alpha value is used depends on other bits in the texel; for details, see "How PIP Tables Work."

The Segment Pointer

Because texel formats that require less than 8 bits per texel do not make full use of the 32-bit entries provided in a PIP table, the design of the PIP table also provides a *segment pointer* that can select from a variable number of segments. For example, a PIP table can be used in 1-bpt (bit-per-texel) mode, in which there are 256 possible segments, or in 8-bpt mode, in which there is only one segment.

By using segments, you can load information for a large number of textures (or variations of textures) in a single PIP table. By using this strategy, you can avoid the need to keep unloading and reloading different PIP tables. To divide a PIP table into segments, you use a shift, a mask and a constant. You apply a shift applied to the input index, and then you use a mask to select between the shifted input index and a constant on a bit-by-bit basis.

What Can Be Stored in a PIP table

Seven bits of alpha information and one control bit (SSB) can also be stored in a PIP table. SSB, alpha, and color can each independently come from any one of these three sources:

- ◆ from the PIP Table (PIP RAM)
- ◆ from a constant
- ◆ from data directly within the source texel.

In order for any of these components to come directly from the source texel, the source texel's format must contain literal data for that component. In other words, if you want to take the alpha component from the source texel, the texture must be of a format that contains either 4 or 7 bits of alpha (see Table 3-4).

You cannot select the source texture as the output of a color value unless the input value for the color was a literal value.

How M2 Interprets Texel Data

If the color data in a texel is interpreted as a PIP offset, resulting in the texel's being passed on to a PIP table, the PIP table can also interpret the texel's color data in several ways, depending on how various texture-related attributes are set. The PIP unit can leave the texel's data values just as they are, or it can look up colors in a color table. It can also treat each of the three components in the texel as a constant. These choices provide great flexibility for certain kinds of effects.

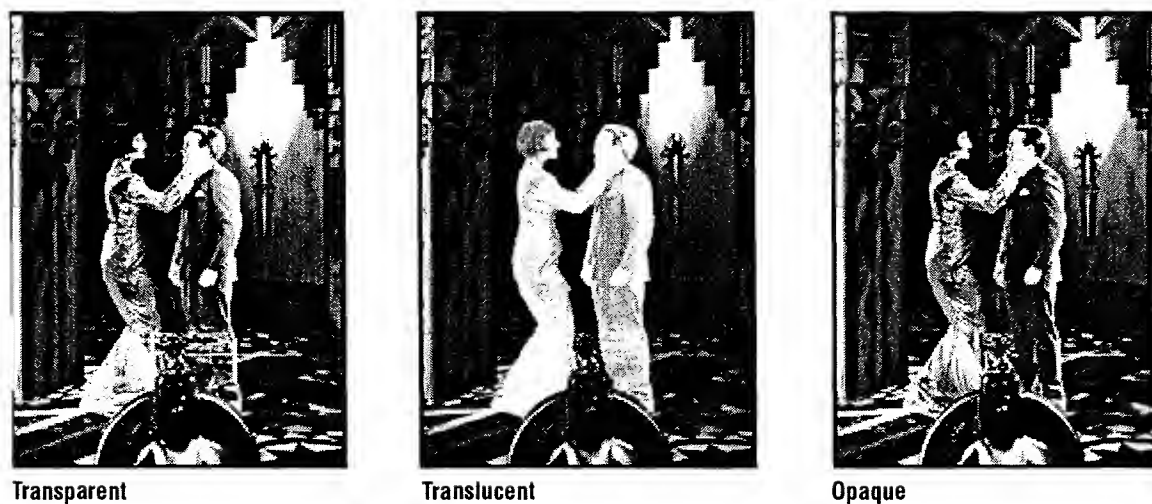


Figure 3-10 *Rendering a ghost*

How PIP Tables Work

The color, alpha, and SSB output of the PIP mapping stage can be obtained from one of three places: it can come from the texel value retrieved from TRAM, from the PIP itself, or from the two constants defined by the user. The utility of PIP mapping can be demonstrated by a simple example.

Let's look at a texture representing a ghost (see Figure 3-10). Assume that the ghost has eight colors and that you want the colors to fade in gradually over a certain number of frames. You can represent this kind of a texture can be represented using four bits per texel —three bits per texel for the color index and one bit per texel for the SSB.

In the texture-mapping example shown in Figure 3-10, the application sets color to come from the PIP table, alpha to come from the constants, and the SSB to come from the TRAM. The alpha component of the first color constant is used for texels

that have an SSB value of 0, and the alpha of the second color constant is used for texels that have an SSB value of 1. The application needs only to change the values of the color constants between different frames to fade out the ghost.

Also, the dynamic range of the alpha value can be up to seven bits. So PIP mapping can be used for certain kinds of animations while achieving significant data compression as well. The output of the PIP mapping stage is used for further processing, as described below.

The color, alpha, and SSB components of `PipConsts` are packed into a `uint32` data type, as shown in Figure 3-11.

SSB	Alpha	Red	Green	Blue
1 bit	7 bits	8 bits	8 bits	8 bits

Figure 3-11 Bit packing of a Color

See Appendix A for an example of the register setup for a PIP load.

Texture Application Blending

The final stage of the texture pipeline is the Texture Application Blender (TAB). This is the stage when the primitive color and alpha components of each texel are blended with the filtered texel value obtained from the filtering stage. Primitive color refers to the color a pixel would have been painted in the absence of texturing. Primitive color is often the result of lighting and shading computations. The blending process for color may be composed of either a LERP or a multiply :

- ◆ LERP— $D = (1 - C) * A + C * B + f(A, B, C)$
- ◆ Multiply— $D = A * B + f(A, B)$

—where A , B , and C are chosen from a list of C_t , A_t , C_{iter} , A_{iter} or constants. $f()$ is an error function. C_t is the color of the texel (from the PIP table or from a source indexed from the PIP table; see “What Can Be Stored in a PIP table” on page 32). A_t is the alpha from the texel. C_{iter} is the iterated color—that is, the interpolated color from the color data at the vertices of the triangle. A_{iter} is the iterated alpha. See Appendix A for an example of how these calculations work.

Multiplying Color and Alpha Components

When you perform LERP and multiplication operations, the input values of A , B , and C are assumed to have a range of $[0, 1]$ —that is, the legal input value includes both 0 and 1. Because A , B , and C are represented by 8-bit numbers, the hexadecimal value `0x00` represents 0 and hex `0xFF` represents 1.0.

Note that in the three application modes described under the following section “Application Modes,” these values result in an error if a simple multiplication operation is performed. For example,

$$0xFF * 0xFF = 0xFE$$

Clearly, the result of this simple multiply should be `0xFF`, not `0xFE`. To compensate for this anomaly, the error function $f()$ is added in. This operation applies an offset to the calculation.

Ideally, this correction should spread out the error term over the full range of output values. However, for ease of implementation, one of the inputs is passed directly to the output if the other input is '1' (0xFF). For the LERP, *A* is passed out directly if *C* is 0x00, and *B* is passed out directly if *C* is 0xFF. For the multiply, *A* is passed out directly if *B* is 0xFF, and *B* is passed out directly if *A* is 0xFF. The case of *A* and *B* both being 0xFF is left as an exercise for the reader.

The only possible function that is applied to the alpha channel is a multiply. Again, this operation passes out an unmodified value if one of the inputs is 1 (0xFF).

Application Modes

Both alpha information and color information can bypass the Texture Application Blender altogether, in which case the output may be either the textured output or the iterated output.

There are three typical application modes: *Modulate*, *Decal* and *Blend*. They are defined as follows:

- ◆ **Modulate**—This is used to generate effects such as a light source illuminating portions of a surface:

$$Cti = Ct * Citer; Ati = At * Aiter$$

- ◆ **Decal**—Here the alpha in the texture map is used to superimpose a texture onto a shaded surface. The texture will not have any effect of lighting. This is useful for effects such as stenciling lettering onto the side of an object:

$$Cti = (1 - At) * Citer + At * Ct; Ati = Aiter$$

- ◆ **Blend**—Here the color information in the texture is used to blend between a shaded surface and a background color. This is useful for effects such as transparency. For example, if we are trying to show the inside of a 3D object, such as a person, the bone would be represented by the constant color in the center and the organs surrounding the bone could be shaded, but with the bone still showing through (i.e. the organs are partially transparent):

$$Cti = (1 - Ct) * Citer + Ct * Cconst; Ati = At * Aiter$$

Texture Mapping Step-by-Step

To boil it all down, M2 texture mapping requires a sequence of three operations:

1. When a particular pixel is to be painted on the screen, texel data that corresponds to the pixel data being used is retrieved from the texture. This texel data value is passed through a PIP module for further mapping. The PIP table may leave the texel data values as they are, perform a color-table lookup, or use a constant for each of the color, alpha, and SSB (source select bit) components. This strategy provides great flexibility for certain kinds of animations.
2. Texture filtering determines the level of detail (LOD) for adjacent pairs of (*U*, *V*) values to reference an appropriate texture map. It performs blending of the color/alpha values from the LODs (a) between adjacent texture maps for linear sampling, or (b) within one LOD for bi-linear sampling, or (c) a mixture of the two for quasi tri-linear sampling.

- Texture application blending applies the texture values obtained after filtering to the color and alpha values of the triangle pixel according to the blend style specified.

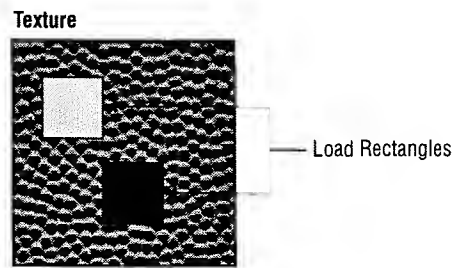


Figure 3-12 *Placing Load Rectangles inside a texture*

Loading Textures

The Texture Mapper contains a DMA engine for loading the Texture RAM (TRAM). For uncompressed textures, the maximum rate at which the TRAM can be loaded is 200MB/sec. The loader is also capable of decompressing a 3DO proprietary compressed texture format that is based around run-length encoding (RLE).

The texture loader performs four main tasks:

- ◆ *MMDMA (Memory to Memory DMA)*—This variety of load can be used to copy a certain number of bytes from main memory into the TRAM or PIP. For textures which do not require special loading operations, such as loading only a sub-rectangle of a texture in RAM into the TRAM, this load mode is the simplest to use.
- ◆ *Uncompressed Texture Load*—This kind of load is used to copy a tile from a source texture within main memory into the TRAM. The source texture must be in uncompressed format. The texel format can be any of the legal M2 formats. For uncompressed textures, a rectangular portion of a texture can be loaded.
- ◆ *Compressed Texture Load*—M2 supports run-length encoded textures. During compressed texture load, a source texture within main memory is decompressed into the TRAM.
- ◆ *PIP Load*—This type of load is used to load the contents of the PIP with a table stored in main memory.

The M2 texture loader is programmed in two stages: by first setting up the loader state and then issuing the TLD command in the TEDCntl register, or using the CLT_TxLoad() macro.

Both these operations can be performed in the command list. Consequently, texture loads can be performed without any overhead from the operating system.

The loader state consists of first selecting the required mode (see the preceding list) within the *TxtLdCntl* register. Each mode requires a different number of loader registers to be setup as well. MMDMA requires only three other registers to be programmed. Compressed texture load requires up to 13 registers to be programmed.

Loading/decompression does not work in parallel with the rest of the Pipeline (that is, the internal hardware pipeline that exists within the Triangle Engine). This is partly because it requires use of the same memory read and write buffers that are used by the destination blender, but it is mainly because the texture load process can swamp the main memory bandwidth while it is in operation. Registers are also shared between the loader and the main pipeline. For correct operation therefore, a SYNC instruction must be placed in the instruction list before any loader setup or the TLD instruction.

Loading Textures by Using MMDMA

MMDMA texture loads require the least amount of runtime setup of the M2 texture loader. In order to use this load mode, however, texture data has to appear in RAM exactly as it should appear in the TRAM. For example, if 3 texture maps with 4 LODs each must be loaded into the TRAM, and all of the texel data from these 12 maps fits within 16K of data, a single MMDMA load can be used to retrieve all 12 textures into the TRAM. To use MMDMA, specify the address in memory of the block of data to DMA in the `TxtLdSrcAddr` register, the byte count to load in the `TxtCount` register, and the word-aligned TRAM load offset in the `TxtLdDstBase` register.

Loading Uncompressed Textures

Uncompressed textures are stored as packed arrays of texels within main memory (see "How Texels Are Stored in Memory"). All the uncompressed texels stored in an array are of the same type, and the M2 hardware can automatically load a smaller tile of the source texture. Each LOD must be stored separately within main memory, and be loaded to a different base address within the TRAM.

Figure 3-13 shows how an array of uncompressed texels is stored in memory.

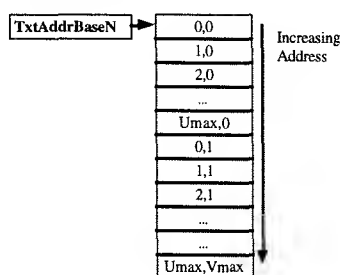


Figure 3-13 An array of uncompressed texels ready for loading

When an application loads an uncompressed texture, the application must point to the start of the first texel to be loaded. To support runtime texture carving (see "Runtime Carving"), and because the source texture can have any of the legal texel depths, the first texel can start on an arbitrary bit boundary. Consequently, the application must specify the bit-aligned address of the first source texel to be loaded (this might not be the same as the start of the texture itself).

The bit address of the first texel to be loaded is the concatenation of `TxtLdSrcBase` and the `SrcBitOff` field from the `TxtLdCntl` register:

```
{TxtLdSrcBase[7:31], SrcBitOff[0:2]}
```

The destination address within the TRAM must be 32-bit word-aligned, and is programmed within the *TxtLdDstBase* register. The source and destination textures can have different strides. These are programmed as bit strides in the *SrcRowBits* and *DstRowBits* fields of the *TxtLdWidth* register. In addition, the total number of rows to be loaded must be programmed in *TxtRowCnt*.

When the TLD instruction is issued, the loader first accesses the byte containing the start texel. The *SrcBitOff* offset is then applied. Next, the loader copies *TxtDstBitCnt* bits into the destination buffer and does the wrapping to re-align the data.

At the end of a line, the source memory pointer is advanced to the start of the next line. This operation is performed by advancing the source pointer by $(\text{TxtSrcBitCnt} - \text{TxtDstBitCnt})$. The operation is performed *TxtRowCnt* times.

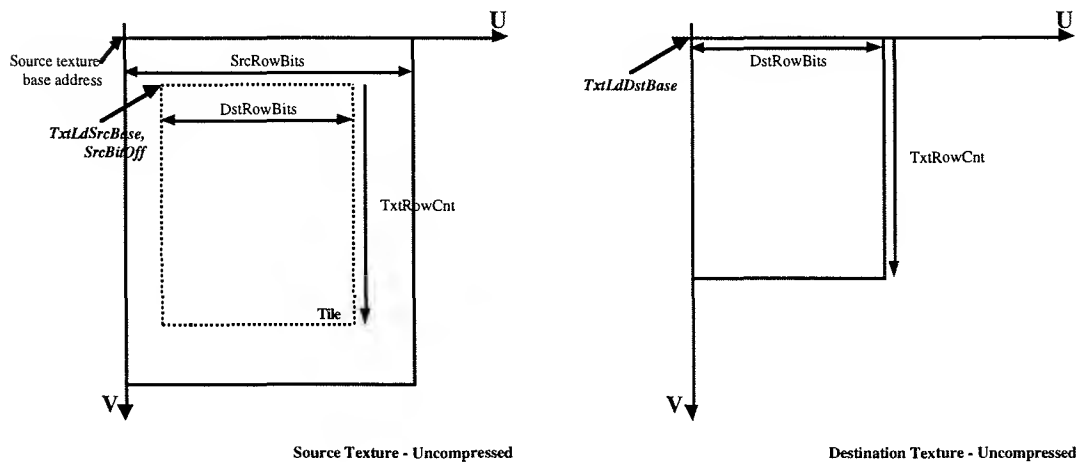


Figure 3-14 *Uncompressed Loader Setup*

See Appendix A for an example of how to load a texture.

Runtime Carving

When a texture is too large to fit within the internal TRAM, the texture (and associated geometry) must be broken into several smaller pieces. This disassembly is supported in hardware by the loader. The application sets the memory base address and also provides a *U* and *V* offset, along with the *V* dimension of the portion to load. This operation allows parts of textures to be loaded automatically.

Although runtime carving is simple in theory, there is one complication you should be aware of. The complication is that bi-linear filtering (as well as tri-linear filtering) require a 2-by-2 array of texels. For texels that are on the edge of a tile, there is no more information present to do the filtering. Hence the tiles must have a *guard region* that is one texel wide around all four edges to allow the joins in tiles to be seamless. When a guard region has been set up, you must ensure that none of the triangle vertices fall within the guard region. For more details about guard regions, see "Mipmap Filtering."

Compressed Texture Load

The M2 texture mapper decompresses textures by reading the control bytes within the compressed texture, and processes the information following each control byte. The processing consists of expanding out run-lengths and resolving different texel formats before placing the data into the TRAM.

The setup for compressed texture load is similar to that of uncompressed loads. The difference is that the texel formats are important during decompress. Also, runtime carving of compressed textures is not supported.

When loading compressed textures, the application provides the base of the compressed texture in `TxtLdSrcBase`, the base of the destination (uncompressed) map in `TxtLdDstBase`, the width of the source and destination in texels within `SrcRowTex` and `DstRowTex`, and the total number of rows to be loaded in `TxtRowCnt`. In addition, the texel formats must be specified as well. This is setup in the four `TxtLdSrcForm` registers and the `TxtExpForm` register. The `TxtLdSrcForm` registers specify the format of the four possible texel types within the source texture in main memory. The `TxtExpForm` register specifies the format of the texels that are to be placed into the TRAM.

See Appendix A for an example showing which registers must be programmed for a compressed texture load.

Texel Expansion

Because multiple texel formats can be present within the source texture, the formats must be expanded out to a common format before you can place them in the output buffer. The desired TRAM texel type is programmed within `TxtExpForm`. All source formats are expanded out to this type. The expansion works by first determining if each channel that is present in the output (SSB, alpha, color) is present in the source. If a certain channel is not present in the source (but is required in the output), then that data is taken from `TxtLdConst[TYPE]`, where `TYPE` is the two-bit index in the control byte. If the data is present in the source, but of a different depth, then the source channel is expanded as described below. The SSB, alpha and color information are handled independently, as follows:

- ◆ *SSB*—If the source texel has an SSB, that SSB is passed through to the expanded texel. If the source texel doesn't have an SSB, but the expanded texel does, a constant SSB is used (`TxtLdConst[TYPE].SSB`)
- ◆ *Alpha*—The source texel can contain 0, 4, or 7 bits of alpha information. If the expanded texel format has more than seven bits of alpha information, the source alpha must be expanded. If the source contains no alpha information, you can expand the source alpha by using a constant alpha (`TxtLdConst[TYPE].alpha`). If the source does contain alpha information, you can expand the source alpha by replicating the MSBs into the LSBs. In the case of constant alpha, either 7 bits or 4 bits are taken from the constant register, depending on the expanded format. Note that for 4-bit output, the top 4 bits of the constant alpha are chosen. Replication can be used only to go from 4 bits to 7 bits of alpha—the top three MSBs of the 4-bit source alpha are replicated into the bottom three LSBs of the expanded alpha.
- ◆ *Literal color*—A literal color can be either 555 or 888. Expansion from 555 to 888 is done by replication of MSBs. If no color is present in the source, then `TxtLdConst[TYPE].red`, `TxtLdConst[TYPE].green`, and `TxtLdConst[TYPE].blue`

are chosen. The top five bits of the constant are chosen for 5-bit output, in much the same way that output is chosen for alpha.

- ◆ *Indexed color*—Source and expanded index depths can be [0, 1, 2, 3, 4, 5, 6, 7, 8] bits. All of these can be expanded to any larger size (up to 8 bits, of course, which is the size of the PIP). The expansion is done by adding an offset (specified in *TxtLdConst[TYPE].index*) and masking to the appropriate depth. If no index is present in the source, then *TxtLdConst[TYPE].index* is used directly.

A special case occurs if the control-bit type is specified as transparent (see "Special Settings in the Control Byte"). In this case, no information follows the control byte. The appropriate color information is provided by one of the constant registers (*TxtLdConst[TYPE]*). Transparency is usually indicated by either zero SSB or alpha. The destination blender is programmed to use either of these values as a transparency flag.

See Appendix A for the register setup used when doing a transparency.

Texture Mapper Pipeline Register Definitions

This section describes the individual registers used to control the texture unit. The section provides descriptions of each of the registers, the address of the register, the CLT macro to set the register and the register layout.

The first half of this section describes the texture mapping registers, which control how a pre-loaded texture is to be interpreted and applied to geometry. The second half describes the registers used to load a texture.

Command Registers, Fields, and Macros

Each command register can be broken down into fields. For example, for a register named *XXX*, the macro *CLA_XXX* provides the data word to be written to the register with the arguments converted and packed into the register fields. The macro *CLT_XXX* takes a pointer to a pointer as the first argument. It writes the argument word to the command list and advances the pointer.

Table 3-5 Register Memory Map

Address	Access	Name	Information
0x000C_0000-0x000C_3FFF	RW	TRAM	Texture RAM
0x0004_6000-0x0004_63FF	RW	PIP	PIP RAM
0x0004_6404	RWSC	TxtLdCntl	Texture Mapper Load Control Register
0x0004_6408	RWSC	TxtAddrCntl	Address Generation Control Register / Filter modes
0x0004_640C	RWSC	TxtPIPCntl	PIP Control Register
0x0004_6410	RWSC	TxtTABCntl	Texture Application Control Register
0x0004_6414	RW	TxtLODBase0	LOD 0 Base Address
0x0004_6418	RW	TxtLODBase1	LOD 1 Base Address
0x0004_641C	RW	TxtLODBase2	LOD 2 Base Address
0x0004_6420	RW	TxtLODBase3	LOD 3 Base Address
0x0004_6424	RW	TxtLdSrcBase TxtMMSrcBase	Texture decompressor source base address MMDMA Source Base
0x0004_6428	RW	TxtCount TxtLdRowCnt TxtLdTexCnt	Byte Count for MMDMA and PIP load Row Count for uncompressed texture load Texel Count for compressed texture load
0x0004_642C	RW	TxtUVMax TxtLdWidth	Texture Size Register Texture Loader Width Register
0x0004_6430	RW	TxtUVMask	Texture Mask Register
0x0004_6434	RWSC	TxtSrcType01	Source Description Registers - Type 0 and 1
0x0004_6438	RWSC	TxtSrcType23	Source Description Registers - Type 2 and 3
0x0004_643C	RWSC	TxtExpType	Expanded Type Description Register
0x0004_6440	RW	TxtConst0 TxtPIPCnst0	Source Expansion Constant Register - Type 0 PIP Constant color - SSB = 0

Address	Access	Name	Information
0x0004_6444	RW	TxtConst1 TxtPIPCConst1	Source Expansion Constant Register - Type 1 PIP Constant color - SSB = 1
0x0004_6448	RW	TxtConst2 TxtTABConst0	Source Expansion Constant Register - Type 2 Texture Application Constant color - SSB = 0
0x0004_644C	RW	TxtConst3 TxtTABConst1	Source Expansion Constant Register - Type 3 Texture Application Constant color - SSB = 1

Warning: Some registers are used for different purposes during the rendering and loading stages.

Texture Generation Registers

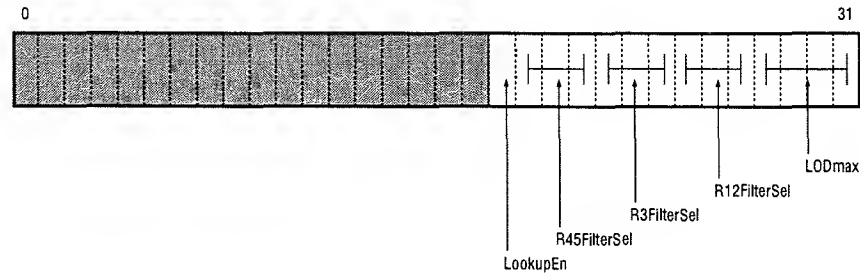
The following paragraphs list and describe the texture units' texture generation registers.

TXADDRCNTL - Address Control Register (0x0004_6408)

The address control register specifies the way in which textures are accessed.

CLT_TXADDRCNTL (pp, textureenable, minfilter, interfilter, magfilter, lodmax)

Field	Description
TEXTUREENABLE	Enables the texture lookup process
MINFILTER	Minification filter modes
POINT BILINEAR	
INTERFILTER	Inter filter modes
POINT LINEAR1 BILINEAR TRILINEAR	
MAGFILTER	Magnification filter modes
POINT BILINEAR	
LODMAX	Number of LODs present -1



The bits shown in the preceding figure are defined as follows:

- ◆ *LookUpEn*—This bit enables the texture lookup process. If the *LookUpEn* bit is disabled, the texture mapper does not generate any stalls to the span walker, irrespective of the texel depth and filter mode. This bit is essentially a user texture enable bit, with one exception: The texture blender is not set up to pass iterated color through by default.
- ◆ *LODmax*—Specifies the number of LODs present.

LODmax[0:3]	Definition
0x0	1 LOD
0x1	2 LODs
0x2	3 LODs
0x3	4 LODs
0x4 - 0xF	reserved

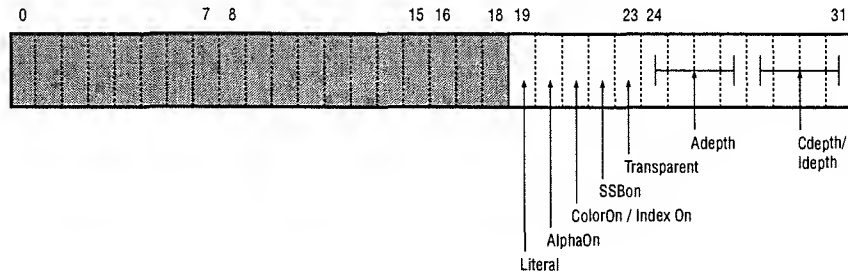
- ◆ *FilterSel*—Specifies the filter modes for the various regions. Each region may have a different filter mode selected.

FilterSel[0:2]	Definition
0x0	Point
0x1	Linear
0x2	Bilinear
0x3	Quasi-Trilinear
0x4 - 0x7	reserved

TXTEXPTYPE - Expansion Type Register (0x0004_643C)

This register describes the texel format within the texture RAM. It is used by the address generation logic to determine the depth of the texels, and by the lookup logic to unpack the data from the TRAM into SSB, color and alpha channels.

```
CLT_TXTEXPTYPE(pp, cdepth, adepth, istrans, hasssb, hascolor,
               hasalpha, isliteral)
```



The bits shown in the preceding figure are defined as follows:

- ◆ *Cdepth*—The number of bits per color component for literal formats. Only a few are valid:

Cdepth	Definition
0x0 - 0x4	reserved
0x5	5bits per color
0x6 - 0x7	reserved
0x8	8bits per color
0x9 - 0xF	reserved

- ◆ *Idepth*—The number of bits of index for indexed formats. Only the following are supported:

Idepth	Definition
0x0	reserved
0x1 - 0x8	1 - 8bits index
0x9 - 0xF	reserved

- ◆ *Adepth*—The number of bits of alpha. Only a few are valid:

Adepth	Definition
0x0 - 0x3	reserved
0x4	4bits alpha
0x5 - 0x6	reserved
0x7	7bits alpha
0x8 - 0xF	reserved

- ◆ *Transparent*—Not used during texture lookup
- ◆ *SSBon*—Specifies whether an SSB is present.

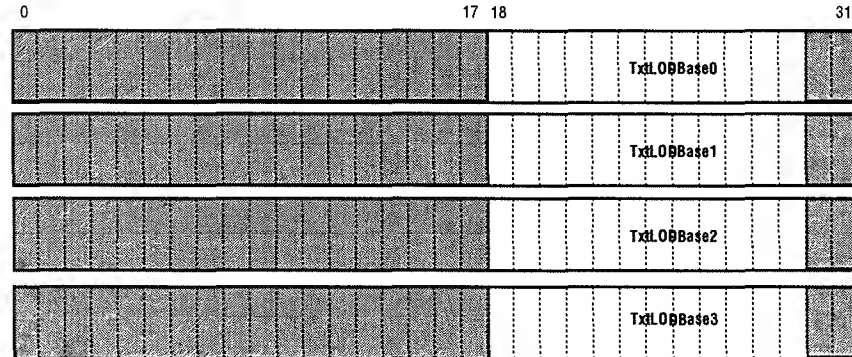
- ◆ *ColorOn, IndexOn*—Specifies whether color is present for literal texel types, or an index is present for indexed types (There are combinations that have no color bits —see below).
- ◆ *AlphaOn*—Specifies whether alpha is present.
- ◆ *Literal*—If set then the type is a literal format. If not set, then the type is indexed. See Table 3-4 for the legal texel values.

TXTL0DBASE0-3 - Texture Base Registers (0x0004_6414 - 0x0004_6420)

The fields *TxtLODBase0* through *TxtLODBase0* shown in the following table are the mipmap base address registers for texture lookup. For correct operation the mipmap addresses should be 32-bit word aligned, i.e. bits 30 and 31 should be zero. This is enforced in hardware for *TxtLODBase1-3*, but not for *TxtLODBase0*, as this register is shared with the loader.

CLT_TXTL0DBASEn (somevalue)

Field	Description
TXTL0DBASE0	Base address for LOD0
TXTL0DBASE1	Base address for LOD1
TXTL0DBASE2	Base address for LOD2
TXTL0DBASE3	Base address for LOD3

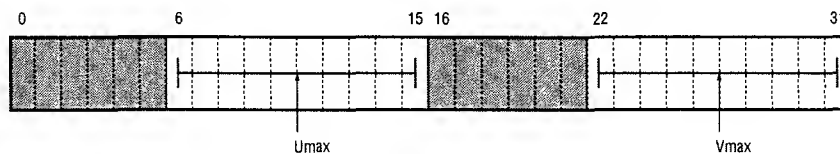


TXTUVMAX - Texture Loader Width Register (0x0004_642C)

The texture size register specifies the size of the decompressed texture during texture lookup. In this case, *UMAX* and *VMAX* (see the following table) specify the limits of the coarsest mipmap present. If *LODmax* is set to 0, this register refers to LOD0. If *LODmax* is set to 3, this register refers to LOD3. *UMAX* and *VMAX* are 10-bit quantities and should be set to the width of the texture (expressed in texels) minus 1.

CLT_TXTUVMAX (pp, umax, vmax)

Field	Description
UMAX	Width of destination buffer in bits
VMAX	Width of source buffer in bits

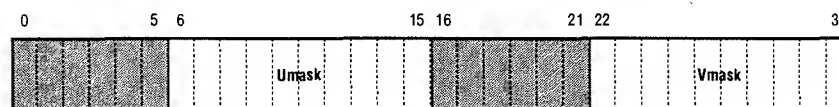


TXTUVMASK - Texture Mask Register (0x0004_6430)

The texture mask is used for texture replication. Bits set to '0' in this register will be masked off during address calculation. For normal operation, this register should be programmed with 0x3FF for both masks. Note that for tiling to work properly as a result, the source texture must be a power of two in U and V. See Appendix A for the register setup used to tile a texture.

CLT_TXTUVMASK (pp, umask, vmask)

Field	Description
UMASK	Enables mask
VMASK	Enables mask

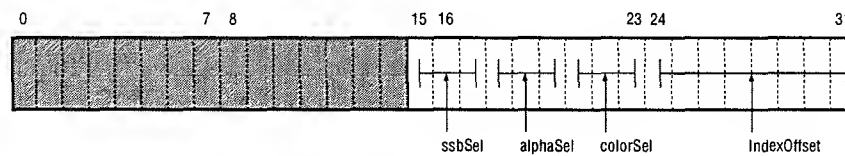


TXPIPCNTL - PIP Control Register (0x0004_640C)

This register controls the operation of the PIP circuitry. For the constant, the input SSB from the TRAM lookup chooses between *TxtPIPConst0* and *TxtPIPConst1* registers. Note that if *colorSel* is 0x1 (meaning that the PIP RAM is bypassed), the SSB or Alpha output should be chosen from either options 0x0 or 0x1 above for correct operation. If there is no SSB in the source texture, the Lookup section passes SSB as 0—so, in this case, *TxtPIPConst0* is always selected.

CLT_TXPIPCNTL (pp, pipssbselect, pipalphaselect, pipcolorselect, pipindexoffset)

Field	Description
PIPSSBSELECT CONSTANT TEXTURE PIP	Select PIP SSB module output
PIPALPHASELECT CONSTANT TEXTURE PIP	Select PIP Alpha module output
PIPCOLORSELECT CONSTANT TEXTURE PIP	Select PIP color module output
PIPINDEXOFFSET	Index offset



The bits shown in the preceding figure are defined as follows:

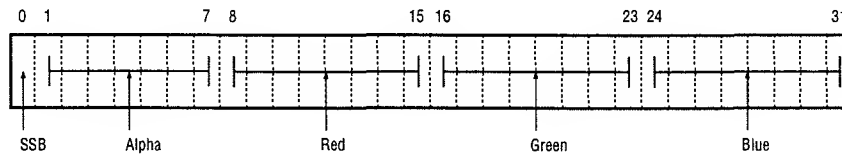
- ◆ *IndexOffset*—This field is added to the input index before accessing into the PIP RAM. This can be seen as a base pointer from where PIP accesses are to occur. The generated index wraps from 255 -> 0 on an overflow.
- ◆ *ssbSel, alphaSel, colorSel*—These fields specify the output of the PIP module for SSB, alpha and color.

ssbSel, alphaSel	Definition
0x0	Constant
0x1	Texture Cache
0x2	PIP
0x3 - 0x7	reserved

PIP Constant Registers (0x0004_6440 - 0x0004_6444)

The SSB from the TRAM Lookup is used to select between these two registers. The output for SSB, alpha or color can be independently controlled to be from the selected register. Note that if no SSB is present in the source texture, then TxtPIPConst0 is always selected.

CLT_TXTCONSTn (pp, blue, green, red, alpha, ssb) n = 0,1



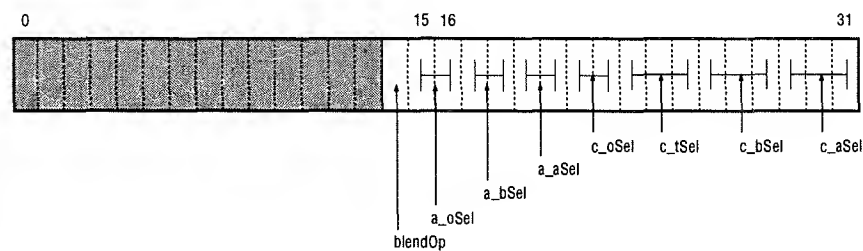
TXTTABCNTL - Texture Application Control Register (0x0004_6410)

This register controls the data path of the texture application blender. It controls the input selects to the color LERP function, and the output selects for the alpha and the color channels independently.

CLT_TXTTABCNTL (pp, firstcolor, secondcolor, thirdcolor, firstalpha, secondalpha, colorout, alphaout, blendop)

Field	Description
FIRSTCOLOR PRIMALPHA PRIMCOLOR TEXALPHA TEXCOLOR CONSTALPHA CONSTCOLOR	Select first color input
SECONDCOLOR PRIMALPHA PRIMCOLOR TEXALPHA TEXCOLOR CONSTALPHA CONSTCOLOR	Select second color input
THIRDCOLOR PRIMALPHA PRIMCOLOR TEXALPHA TEXCOLOR CONSTALPHA CONSTCOLOR	Select third color input
FIRSTALPHA PRIMALPHA TEXALPHA CONSTALPHA	Select first alpha input
SECONDALPHA PRIMALPHA TEXALPHA CONSTALPHA	Select second alpha input

Field	Description
COLOROUT PRIMCOLOR TEXCOLOR BLEND	Select color output
ALPHAOUT PRIMALPHA TEXALPHA BLEND	Select Alpha output
BLENDOP	Control the LERP function



The bits shown in the preceding figure are defined as follows:

- ◆ *c_aSel, c_bSel, c_tSel*—Control the inputs to the color LERP function.

<i>c_aSel, c_bSel, c_tSel</i>	Definition
0x0	Aiter
0x1	Citer
0x2	At
0x3	Ct
0x4	Aconst
0x5	Cconst
0x6 - 0x7	reserved

Note: The constant refers to the two constant registers: *TxtCATIConst0* and *TxtCATIConst01*. They are selected by the SSB coming into the blender from the filter.

- ◆ *a_aSel, a_bSel*—Control the inputs to the alpha Multiplier function.

<i>a_aSel, a_bSel</i>	Definition
0x0	Aiter

a_aSel, a_bSel	Definition
0x1	At
0x2	Aconst
0x3	reserved

- ◆ *c_oSel, a_oSel*—These fields control the datapath mux at the output from the blender. The two are independent.

c_oSel	Definition
0x0	Citer
0x1	Ct
0x2	Blend output
0x3	reserved

a_oSel	Definition
0x0	Aiter
0x1	At
0x2	Blend output
0x3	reserved

Note: If options 0 or 1 are selected above, then the inputs to the LERP block and alpha multiply are don't care.

- ◆ *blendOp*—Controls the function of the LERP.

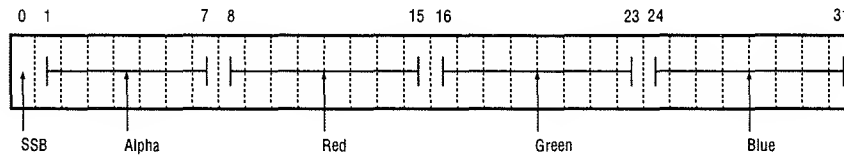
blendOp	Definition
0x0	LERP ($A*(1-t) + B * t$)
0x1	MULT ($A * B$)

Note: For the MULT function, the t input to the LERP block is don't care.

TAB Constant Registers (0x0004_6448 & 0x0004_644C)

The SSB from the output of the filter is used to select between these two registers. The output for SSB, alpha or color from the TAB can be independently controlled to be from the selected constant register or from the pipeline.

CLT_TXTCONSTn (pp, blue, green, red, alpha, ssb) n = 2, 3



Texture Loader Register Definitions

The texture loader registers are defined in the following paragraphs.

TXTCONST (0, 1, 2, 3) - Source Expansion Constant Registers (0x0004_6440 - 0x0004_644C)

These four sets of constants registers are used during the decompression of compressed textures to generate constant colors or index for each of the four source texture texel types. TXTCONST0 and TXTCONST1 select the PIP constant. TXTCONST2 and TXTCONST3 select the texture application constant. For indexed types, the *Blue* field can be used as an offset that is added to the input index value. If no index value is provided, then several bits (depending on the color depth) may be taken as an index constant.

Note that when 5 bits of color is selected for expansion, the 5-bit color is right-aligned into the 8-bit channel. Likewise, 4-bit alpha is right-aligned into the 7-bit alpha channel.

The SSB from the TRAM Lookup is used to select between these two registers. The output for SSB, alpha or color can be independently controlled to be from the selected register.

CLT_TXTCONSTn (pp, blue, green, red, alpha, ssb)

Field	Description
BLUE	Select Blue value
GREEN	Select Green value
RED	Select Red value
ALPHA	Select Alpha value
SSB	Select SSB value

TXTSRCTYPE n - Texture Source Types (0x0004_6434 - 0x0004_643c)

These register are used during texture load of compressed textures only. There are five format description registers—one for each of the four compressed texel types within the source texture (0, 1, 2, 3), and one for the expanded format that is used to be loaded into the TRAM.

CLT_TXTSRCTYPE n (pp, cdepth n , adepth n , istrans n , hassssbn, hascolor n , hasalphan, isliteral n)

Field	Description
CDEPTH(0, 1, 2, 3)	Number of bits per color component for literal formats (0x5=5 bits per color, 0x8=8 bits per color)
ADEPTH(0, 1, 2, 3)	Number of Alpha bits (0x4=4 bits Alpha, 0x7=7 bits)
ISTRANS(0, 1, 2, 3)	Selects constant Alpha or SSB for run of compressed texels
HASSSB(0, 1, 2, 3)	Specifies if SSB is present
HASCOLOR(0, 1, 2, 3)	Specifies if color is present for literal texel types
HASALPHA(0, 1, 2, 3)	Specifies if Alpha is present
ISLITERAL(0, 1, 2, 3)	If set, the type is a literal format

CLT_TXTEXPTYPE(pp, cdepth, adepth, istrans, hasssb, hascolor, hasalpha, isliteral)

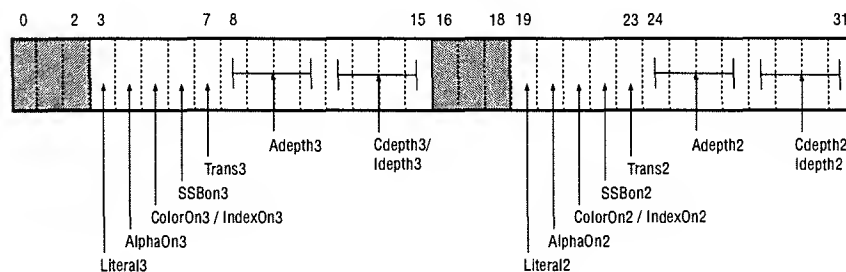
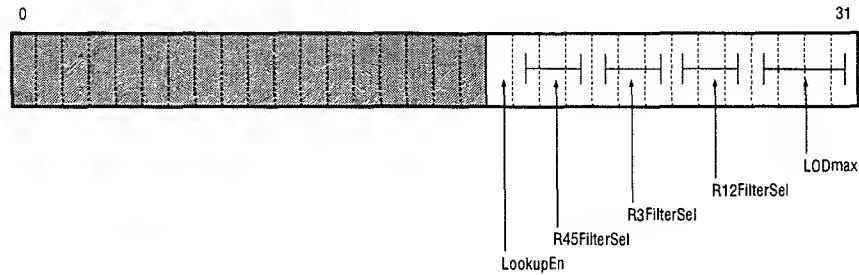
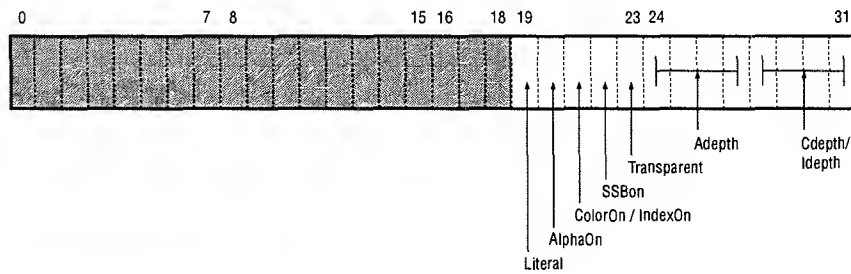


Figure 3-15 *TxtLdSrcType01*

Figure 3-16 *TxtLdSrcType23*Figure 3-17 *TxtExpType*

These registers are used during texture load of compressed textures only. There are five format description registers—one for each of the four compressed texel types within the source texture, and one for the expanded format that is used to be loaded into the TRAM.

The fields specify the following information:

- ◆ *Cdepth*—Literal formats are not supported in compressed textures.
- ◆ *Idepth*—The number of bits of index for indexed formats. Only the following are supported:

Idepth	Definition
0x0	reserved
0x1 - 0x8	1 - 8bits index
0x9 - 0xF	reserved

- ◆ *Adepth*—The number of bits of alpha. Only a few are valid:

Adepth	Definition
0x0 - 0x3	reserved
0x4	4 bits alpha
0x5 - 0x6	reserved

Adepth	Definition
0x7	7 bits alpha
0x8 - 0xF	reserved

- ◆ *Transparent*—Used only for compressed texture source description. If this field is set, the decompressor uses a constant color, alpha or SSB for that run length of compressed texels. The constant register is selected from one of the four *TxtSrcConstN* registers, where *N* indicates the type of the current control byte within the decompressor.
- ◆ *SSBon*—Specifies whether an SSB is present.
- ◆ *ColorOn, IndexOn*—Specifies whether index color is present. Note that there are combinations that have no color bits.
- ◆ *AlphaOn*—Specifies whether alpha is present.
- ◆ *Literal*—If this field is set, the type is a literal format. If this field is not set, the type is indexed.

The decompressor can support all the indexed pipeline formats. Literal pipeline formats can not be compressed. The texels are decompressed at different rates depending on their total depth. The following table shows the complete set of supported texels and the speed at which they can be loaded.

Color	Alpha	SSB	Total	Speed
0	-	1	1	4
1	-	-	1	4
1	-	1	2	4
2	-	-	2	4
0	4	-	4	4
3	-	1	4	4
4	-	-	4	4
6	-	-	6	2
5	-	1	6	2
1	4	1	6	2
2	4	-	6	2
0	7	1	8	2
3	4	1	8	2
4	4	-	8	2
7	-	1	8	2
8	-	-	8	2

Color	Alpha	SSB	Total	Speed
4	7	1	12	1
7	4	1	12	1
8	4	-	12	1
8	7	1	16	1

In the preceding table, the speed column specifies the number of texels loaded per tick. The speed is independent of the source type—it depends only on the expanded type to be written to the TRAM.

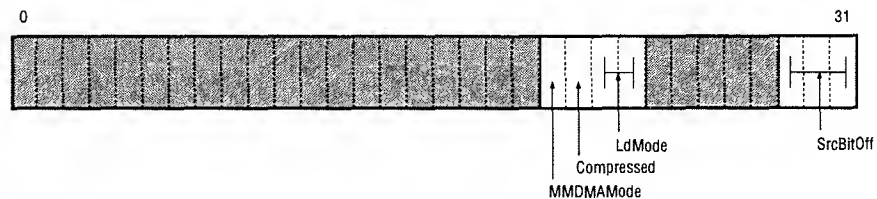
Note also that compressed source texels cannot contain more data for each of the data types (color, alpha, SSB) than the expanded format. They can only contain an equal amount or less (none).

TXTLDCNTL - Texture Loader Control Register (0x0004_6404)

This register controls the texture load process. When the TLD instruction is issued, this register is used to determine which type of load function to select.

CLT_TXTLDCNTL (pp, compressed, loadmode, srcbitoff)

Field	Description
COMPRESSED	Specifies that the source texture in memory is compressed and needs decompression before use.
LOADMODE LOADMODE_TEXTURE LOADMODE_MMDMA LOADMODE_PIP	Determines the operation of the memory-to-memory DMA process.
SRCBITOFF	Source bit offset used with uncompressed texture loading.



The bits shown in the preceding figure are defined as follows:

- ◆ *LdMode*—Specifies the mode of the texture loader.

LdMode[0:1]	Definition
0x0	Texture Load
0x1	MMDMA
0x2	PIP Load
0x3	reserved

Note: When MMDMA is selected, the target is determined by the MMDMATramOn and MMD-MAPIpOn bits within the supervisor TxtCntl register.

- ◆ *Compressed*—If this bit is enabled, it indicates that the source texture in memory is compressed and must be decompressed before it is used. This bit is used only if *LdMode* = 0 (i.e., Texture Load).
- ◆ *SrcBitOff*—This field is used by the texture loader during the loading of uncompressed textures (*LdMode* = 0 & *Compressed* == 0). It indicates the bit start position within a byte. This is used in conjunction with *TxtLdSrcBase* to define the exact bit start position of a texel in memory.

TXTLDSRCOFFSET - Text Load Source Offset Register

This register provides the compressed texture start offset value.

CLT_TXTLDSRCOFFSET (somevalue)

Field	Description
VALUE	Select offset value

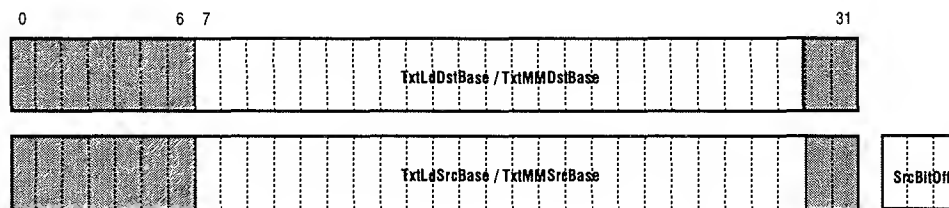
TXTLDSRCADDR- Loader Base Registers(0x0004_6424 & 0x0004_6414)

TxtLdSrcBase describes the byte-aligned source address in main memory. Compressed textures may start on any byte aligned address. Uncompressed textures may start on any bit address. The *SrcBitOff* field within the *TxtLdCntl* register is effectively used as three extra MSBs for uncompressed load. PIP contents are byte aligned as well as MMDMA source addresses.

CLT_TXTLDSRCADDR(pp, x)

TxtLdDstBase describes the 32-bit word aligned destination address within the TRAM. The base address of all load functions will be on a 32-bit boundary. This is because the TRAM logic does not support a RMW cycle into the RAM to allow for arbitrary alignment

CLT_TXTLODBASE0(pp, x)



TXTCOUNT - Texture Loader Count Register (0x0004_6428)

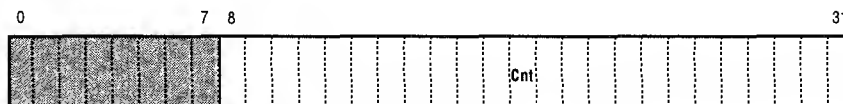
This register specifies different counts for the various loader modes:

- ◆ During uncompressed texture load, TxtLdRowCnt indicates the number of rows of texels to copy into the TRAM.
- ◆ During compressed texture load, TxtLdTexCnt indicates the total number of texels to decompress.
- ◆ During MMDMA, TxtLdByteCnt indicates the number of bytes to copy.
- ◆ During PIP load, TxtLdByteCnt indicates the number of bytes to copy into the PIP.

CLT TXTCOUNT (somevalue)

Field	Description
TXTCOUNT	<p>During compressed texture load, indicates the total number of texels to decompress.</p> <p>During compressed texture load, indicates the total number of texels to decompress.</p> <p>During MMDMA, indicates the number of bytes to copy.</p> <p>During PIP load, indicates the number of bytes to copy into the PIP.</p>

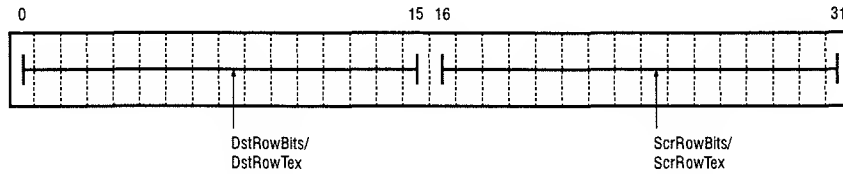
Note that a count of 0 does mean zero—that is, “Do something zero times.”



TxtUVMax-Texture Loader Width Register (0x0004_642C)

During texture load of uncompressed textures, this register indicates the width of the destination and source buffers in bits. A count of 0 indicates a width of zero.

CLT_TXTUVMAX(pp, umax, vmax)

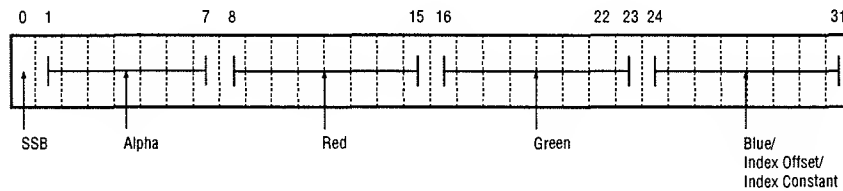


TxtConst2-3-Decompress Constant Registers (0x0004_6448 - 0x0004_644C)

M2 provides two constant registers. They are used only during decompresses of compressed textures. They are used to generate constant colors or index for each of the four source texture texel types. For indexed types the *Blue* field can be used as an offset which is added to the input index value. If no index value is provided, then several bits (depending on the color depth) may be taken as an index constant.

`CLT_TXTCONSTn(pp, red, green, blue, alpha, ssb) n = 2,3`

Note that when 5 bits of color are selected for expansion, the 5-bit color is right-aligned into the 8-bit channel. Likewise, 4-bit alpha is right-aligned into the 7-bit alpha channel.



Destination Blender

Once pixels have been passed through the Texture Mapper, they pass into the Destination Blender, where the last four processing steps take place:

- Pixels can be optionally discarded from a triangle, based on several criteria.
- Pixels can be blended with a color, or pixels from another frame buffer.
- Pixels can be dithered.
- Pixels can be Z-buffered, so that they are only drawn if they are from visible portions of a triangle that intersects other triangles.

Finally, the pixels are written to the frame buffer.

This chapter contains the following topics:

Topic	Page Number
Pixel Discards	60
Pixel Blending	60
Pixel Scaling	61
Source Blending	61
Dithering	62
Z-buffering	62
Window Clipping	63
Destination Blender Register Definitions	63

Pixel Discards

In the Destination Blender, pixels can be discarded before being output to the frame buffer or Z-buffer. This can be useful, for example, to mask away unwanted parts of a triangle that fall outside of the shape of a sprite that is texture-mapped onto a pair of triangles. Discards can be set to happen whenever any combination of the following events occur:

- Alpha of the pixel is 0.0
- Color of the pixel is pure black (red=0.0, green=0.0, blue=0.0)
- SSB of the pixel is 0
- Pixel falls outside the area covered by the Z-buffer

Note: *Since the pixels enter the Destination Blender after being passed through the Texture Mapper, the color, alpha, and SSB used for discards are the ones that come from the Texture Blender.*

The Triangle Engine only allows you to allocate a Z-buffer that is smaller than the frame buffer. The Z-discard allows pixels that fall outside of the Z-buffered area to be immediately discarded. For more information, see the “Z-buffering” section, later in this chapter.

There is no performance hit when discards are enabled, and they can allow better performance in some cases, since discarded pixels are not passed into the Z-buffering unit or pixel blender.

Pixel Blending

After the discard process is complete, the remaining pixels are passed to a sophisticated blender, where the color can be blended with a color from any of a number of sources, including a second source frame buffer. This blender is typically used to make pixels partially transparent, or to globally tint a scene toward a constant color, such as in a screen fade. Because the blend can be weighted based on alpha, carefully constructed triangles can use the Destination Blender to apply fog and some other special effects to the scene.

The two colors to be blended (we’re calling them A and B) can be blended according to the following equation:

$$C_{out} = ((MA \bullet A) \text{ op } (MB \bullet B)) << \text{shift}$$

A, B, MA, and MB can come from a variety of sources. The following table describes the different combinations that can be used for these variables:

Variable	Possible Sources
A	Color of the pixel from the Texture Mapper Constant color Alpha of the pixel from the Texture Mapper Complement of the color from the source frame buffer

Variable	Possible Sources
B	Color from a source frame buffer Constant color Alpha of the pixel from the Texture Mapper Complement of the color of the pixel from the Texture Mapper
MA	Alpha of the pixel from the Texture Mapper Alpha from the source frame buffer One of a pair of constants (selected by the SSB) Color of the pixel from the source frame buffer
MB	Alpha of the pixel from the Texture Mapper Alpha from the source frame buffer One of a pair of constants (selected by the SSB) Color of the pixel from the Texture Mapper

Besides the flexibility that the various selectors for A, B, MA, and MB provide, you can also decide what operation you want to perform. The operation can be an add or subtract, with or without clamping, or any boolean operator, such as AND, OR, etc.

Finally, color components can be bit-shifted left or right up to 3 bits. The shift is actually always a left shift, but the parameter allows for a shift left by -3 bits, which results in a right shift 3 bits.

When the pixel blender is enabled, pixels can be output at a maximum throughput of 1 pixel per tick.

Pixel Scaling

The Triangle Engine performs color calculations on 8-bit numbers for each channel. Source images, as well as destination frame buffers, can be 16- or 32-bit numbers. The Triangle Engine converts the 16-bit pixel values to 32-bit values before any computations are performed.

You have an option to right-shift by three bits the input source and/or texture 8-bit values. In this way, your application can combine as many as eight images without overflowing the pixel values. You can use this capability to accumulate images in the destination frame buffer. Resulting accumulated images are then scaled appropriately. Attributes associated with this capability are the `DblARightJustify` attribute, the `DblBRightJustify` attribute, and the `DblFinalDivide` attribute.

Source Blending

By blending pixels with a source frame buffer, you can easily produce effects like transparency and reflections. But, you need to set up the correct registers in order to ensure correct output. The M2 Triangle Engine needs to know the source frame buffer's address, width, height, pixel depth, and stride. The source frame buffer can be the same bitmap that is used as the destination frame buffer.

When blending with a source frame buffer is enabled, pixels can be output at a maximum rate of 1 pixel every other tick.

Dithering

The M2 Triangle Engine can dither pixels before they are sent to the destination frame buffer. This reduces the visual artifact of mock banding when 16-bit frame buffers are used. Here's how the ditherer works:

You load a 4 x 4 matrix of dither values. Valid matrix values are 4-bit signed integers, between -8 and +7. The matrix is packed into two 32-bit registers. One register contains the matrix values for the top two rows, the other the bottom two rows.

The least significant two bits of the x- and y-coordinates of the pixel are used as indices into the 4 x 4 matrix. The value from the matrix is retrieved, and added to the red, green, and blue components of the pixel.

Dithering can be performed at a rate of 2 pixels per tick.

See Appendix A for the register setup used with dithering.

Z-buffering

The M2 Triangle Engine supports Z-buffering for pixels that contain depth information (triangles which have a W value). By using a Z-buffer, you can render objects in any order, and have them intersect each other, without worrying about how overlapping regions are drawn. Normally, the Z-buffer only draws pixels that are closer to the viewer than previously-rendered pixels, but it can be set to work in other modes.

The Z-buffer works by storing depth information in a special 16-bit frame buffer. The contents of each 16-bit pixel are in a proprietary 3DO format. At the beginning of a frame, the best way to clear the Z-buffer is to render two large triangles with a W value of 0.0, and set the Z-bufferer to force updates to the Z-buffer, regardless of what the value is. For more information about modifying the operating mode of the Z-buffer, see the description of the DBZCNTL register in the "Destination Blender Register Definitions" section, later in this chapter.

It is not necessary to allocate a Z-buffer to be the size of the entire frame buffer. In some cases, such as an application where the top portion of the screen is used for player's status, it might only be necessary to allocate a Z-buffer to be the size of the playfield area of the screen, thus saving some memory.

Note: *Although the Z-buffer does not need to be as tall as the destination frame buffer, it must be as wide as the destination frame buffer's stride.*

Z-banding

One other technique that can save some RAM (but at the cost of some performance) is to allocate a Z-buffer that is a fraction of the height of the frame buffer. This technique is referred to as *Z-banding*. Here's how Z-banding works: The Z-buffer is aligned with the top of the destination frame buffer. The scene is rendered and the Z-buffer is moved down. The scene is re-rendered and the Z-buffer moved down again. This process is repeated until the whole frame buffer is drawn. Using this technique, in combination with the Destination Blender's discard when pixels are outside the region covered by the Z-buffer, allows Z-buffering to occur when there is not enough RAM for an entire Z-buffer.

However, because the Triangle Engine needs to pass over the data multiple times, performance will usually not be as good as the case where a complete Z-buffer can be allocated.

When Z-buffering is enabled, performance is 2 pixels per tick if the pixels do not need to update the frame buffer or Z-buffer, and 1 pixel per tick in the case where the pixel needs to be drawn.

See Appendix A for the register setup used with Z-buffering.

Window Clipping

An application can specify a rectangular window for clipping out the pixels that lie on the inside or the outside of this rectangle. The rectangle should be specified in conjunction with the `Db1EnableAttrs` attributes of `WinClipInEnable` or `WinClipOutEnable`. The *wWindow* rectangle is specified with `Db1XWinClipMin`, `Db1XWinClipMax`, `Db1YWinClipMin`, and `Db1YWinClipMax`.

Destination Blender Register Definitions

This section describes the individual registers used to control the Destination Blender. Each command register can be broken down into fields. For a register named XXX, the macro `CLA_XXX` provides the data word to be written to the register with the arguments converted and packed into the register fields. The macro `CLT_XXX` takes a pointer to a pointer as the first argument. It writes the argument word to the command list and advances the pointer.

This section provides descriptions of each of the registers, the address of the register, the CLT macro to set the register and the register layout.

Warning: *Some registers are used for different purposes during the rendering and loading stages.*

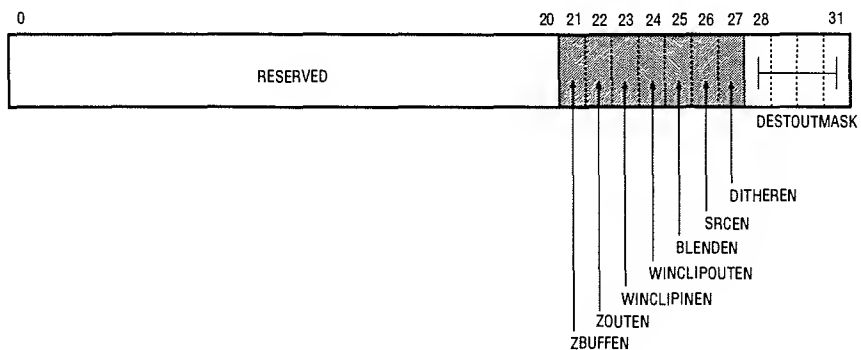
DBUSERCONTROL - User Destination Blend Control Register (0x0004_8008)

This register provides general control of the destination blend logic.

`CLT_DBUSERCONTROL(pp, zbuffen, zouten, winclipinen, winclipouten, blenden, srcen, ditheren, destoutmask)`

Field	Description
ZBUFFEN	Enable Z-buffering.
ZOUTEN	Enable Z-buffer output.
WINCLIPINEN	Enable window clipping inside the clip range.
WINCLIPOUTEN	Enable window clipping outside the clip range.
BLENDEEN	Enable color, Alpha, and DSB blending.
SRCEN	Master control for the source input. Enables color, Alpha, and DSB blending.

Field	Description
DITHEREN	Enable dithering for output to 555.
DESTOUTMASK	Enable byte output (DSB, RGB, Alpha)

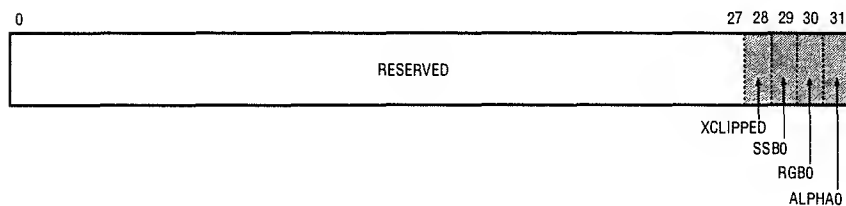


DBDISCARDCONTROL - Pixel Discard Control Register (0x0004_800c)

This register provides pixel discard control.

CLT_DBDISCARDCONTROL (pp, zclipped ssb0, rgb0, alpha0)

Field	Description
ZCLIPPED	Enable pixel discard when outside of the buffer region.
SSB0	Enable pixel discards based on SSB.
RGB0	Enable pixel discards if R == G == B == 0.
ALPHA0	Enable pixel discards if Alpha == 0.



DBINTCNTL - Interrupt Control Register (0x0004_8014)

This register provides masks to control which ALU and Z results can generate

interrupts to the CPU.

CLT_DBINTCNTL (PP, ALUSTAT, ZFUNCSTAT)

Field	Description
ALUSTAT	AND'd with ALU status bits and OR'd to create ALUStatInt
ZFUNCSTAT	AND'd with Z function status bits and OR'd to create ZFuncInt

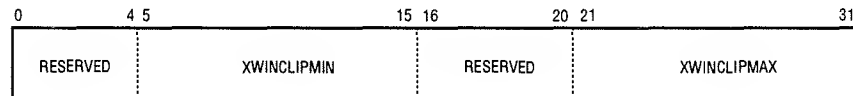


DBXWINCLIP - X Window Clip Value Register (0x0004_801c)

This register defines the X border of the user-defined render window.

CLT_DBXWINCLIP (pp, xwinclipmin, xwinclipmax)

Field	Description
XWINCLIPMIN	X window clip minimum (left)
XWINCLIPMAX	X window clip maximum (right)

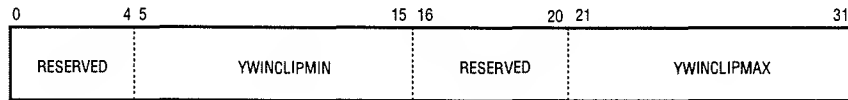


DBYWINCLIP - Y Window Clip Value Register (0x0004_8020)

This register defines the Y border of the user-defined render window.

CLT_DBXWINCLIP (pp, xwinclipmin, xwinclipmax)

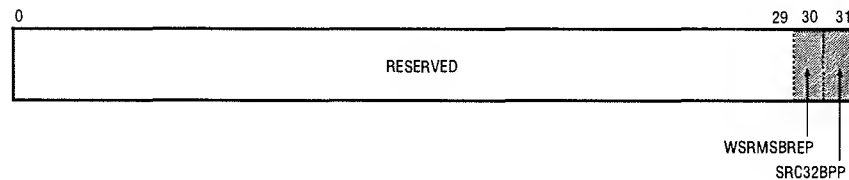
Field	Description
YWINCLIPMIN	Y window clip minimum (left)
YWINCLIPMAX	Y window clip maximum (right)

**DBSRCCNTL - Source Read Control Register (0x0004_8030)**

Specifies the format of the buffer to be used for source blending.

CLT_DBSRCCNTL (pp, srcmsbred, src32bpp)

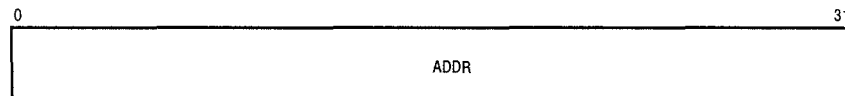
Field	Description
WSRMSBREP	For 16 Bpp, Replicate 3 MSBs into 3 LSB's if left justified (otherwise insert zeros).
SRC32BPP	Source pixel format (16 or 32 bpp).

**DBSRCBASEADDR - Source Base Address Register (0x0004_8034)**

This register provides the source bitmap address when the destination blender's source blending mode is enabled.

CLT_DBSRCADDR (pp, addr)

Field	Description
ADDR	Base address of source buffer

**DBSRCXSTRIDE - Source X Stride Register (0x0004_8038)**

This register contains the source read X stride value - the distance in pixels to the

next line.

```
CLT_DBSRCXSTRIDE (pp, xstride)
```

Field	Description
XSTRIDE	X Stride value.

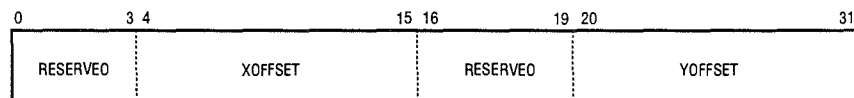


DBSRCOFFSET - Source Offsets Register (0x0004_803c)

This register enables the X and Y offsets used for source reads.

```
CLT_DBSRCOFFSET (pp, xoffset, yoffset)
```

Field	Description
XOFFSET	X offset
YOFFSET	Y offset

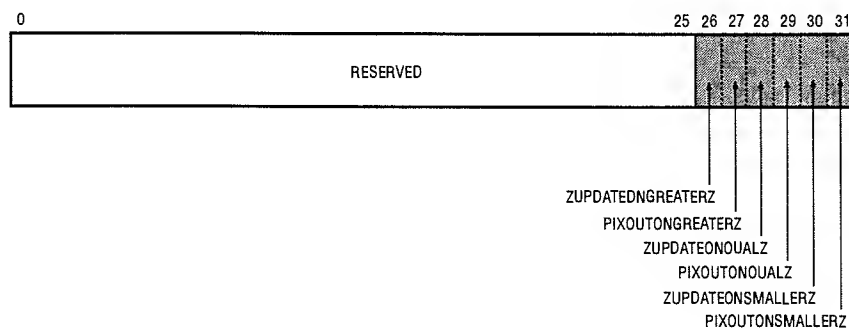


DBZCNTRL - Z Buffer Control Register (0x0004_8040)

This register defines the ZPixOut and ZBufOut results.

```
CLT_DBZCONTROL (pp, zupdateongreaterz, pixoutongreaterz,
                zupdateonqualz, pixoutonqualz, zupdateonsmallerz,
                pixoutonsmallerz)
```

Field	Description
ZUPDATEONGREATERZ	If new Z greater than current Z, update Z
PIXOUTONGREATERZ	If new Z greater than current Z, update pix
ZUPDATEONQUALZ	If new Z equal to current Z, update Z
PIXOUTONQUALZ	If new Z equal to current Z, update pix
ZUPDATEONSMALLERZ	If new Z smaller than current Z, update Z
PIXOUTONSMALLERZ	If new Z smaller than current Z, update pix

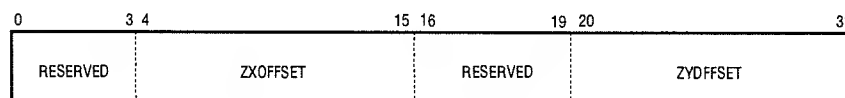


DBZOFFSET - Z Offset Register (0x0004_8048)

This register contains the X and Y offsets used for Z.

CLT_DBZOFFSET (pp, zxoffset, zyoffset)

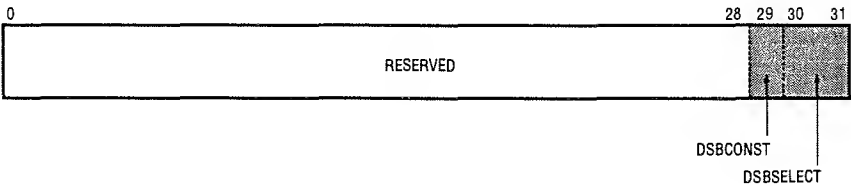
Field	Description
ZXOFFSET	X offset for Z
ZYOFFSET	Y offset for Z



DBSSBDSBCNTL - SSB/DSB Control Register (0x0004_8050)

CLT_DBSSBDSBCNTL (pp, dsconst, dbselect_XXXXXXX)

Field	Description
DSBCONST	DSB constant
DSBSELECT_	Select DSB generation:
OBJSSB	Use SSB (0)
CONST	Use constant (1)
SRCSSB	Use source input (2)

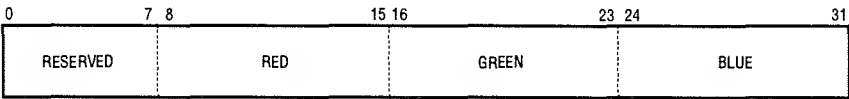


DBCONSTIN - Constant In Register (0x0004_8054)

This register contains the RGB constant used for computation input.

CLT_DNCONSTIN (pp, red, green, blue)

Field	Description
RED	Red constant
GREEN	Green constant
BLUE	Blue constant



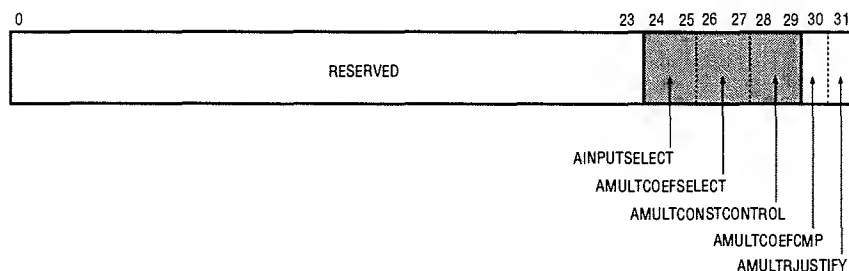
DBAMULTCNTL - Texture Multiplication Control Register (0x0004_8058)

This register controls the texture color multiplication.

CLT_DBAMULTCNTL (ainputselect, amultcoefselect,
amultconstcontrol, amultrjustify)

Field	Description
AINPUTSELECT	Choose texture, constant, or 1-Cs for texture input: TEXCOLOR (0) CONSTCOLOR (1) SRCCOLORCOMPLEMENT (2) TEXALPHA (3)

Field	Description
AMULTCOEFSELECT	Choose texture coefficient: TEXALPHA (00) SRCALPHA (08) CONST (10) SRCCOLOR (18) TEXALPHACOMPLEMENT (01) SRCALPHACOMPLEMENT (09) CONSTCOMPLEMENT (11) SRCCOLORCOMPLEMENT (19)
AMULTCONSTCONTROL	Choose constant controller: TEXSSB (0) SRCSSB (1)
AMULTCOEFCMP	Use (1-coef) for texture coefficient
AMULTRJUSTIFY	Right shift 888 texture values to 555

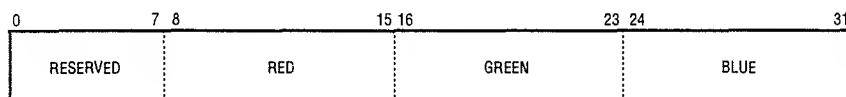


DBAMULTCONSTSSB0 - Texture Coefficient Constant 0 Register (0x0004_805c)

This register contains the texture RGB coefficient constant "0."

CLT_DBAMULTCONSTSSB) (pp, red, green, blue)

Field	Description
RED	Red constant 0
GREEN	Green constant 0
BLUE	Blue constant 0

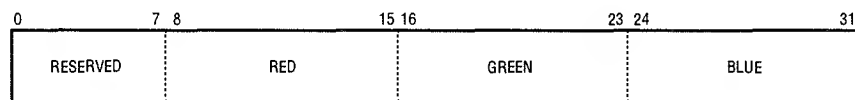


DBAMULTCONSTSSB1 - Texture Coefficient Constant 1 Register (0x0004_8060)

This register contains the texture RGB coefficient constant “1.”

CLT_DBAMULTCONSTSSB1 (pp, red, green, blue)

Field	Description
RED	Red constant 1
GREEN	Green constant 1
BLUE	Blue constant 1

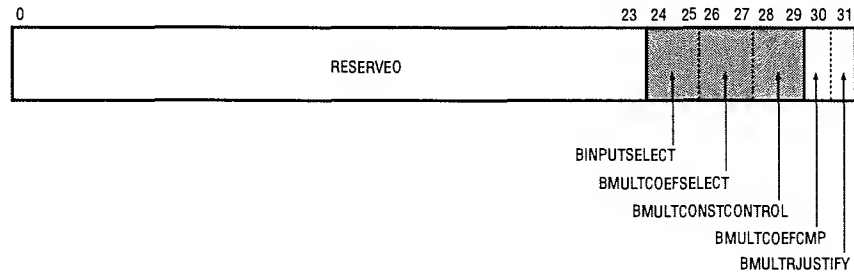
**DBBMULTCNTL - Source Multiplication Control Register (0x0004_8064)**

This register provides the source multiplication control.

CLT_DBBMULTCNTL (pp, binputselect, bmultcoefselect, bmultconstcontrol, bmultrjustify)

Field	Description
BINPUTSELECT	Choose source or constant for source input: SRCCOLOR (0) CONSTCOLOR (1) TEXCOLORCOMPLEMENT (2) SRCALPHA (3)
BMULTCOEFSELECT	Choose source coefficient: TEXALPHA (00) SRCALPHA (08) CONST (10) TEXCOLOR (18) TEXALPHACOMPLEMENT (01) SRCALPHACOMPLEMENT (09) CONSTCOMPLEMENT (11) TEXCOLORCOMPLEMENT (19)
BMULTCONSTCONTROL	Choose constant controller: TEXSSB (0) SRCSSB (1)

Field	Description
BMULTCOEFCMP	Use (1-coef) for source coefficient
BMULTRJUSTIFY	Right shift 888 source values to 555

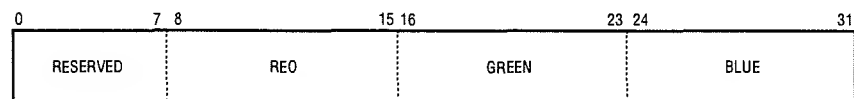


DBBMULTCONSTSSB0 - Source Coefficient Constant 0 Register (0x0004_8068)

This register contains the source RGB coefficient constant "0."

CLT_DBBMULTCONSTSSB0 (pp, red, green, blue)

Field	Description
RED	Red constant 0
GREEN	Green constant 0
BLUE	Blue constant 0

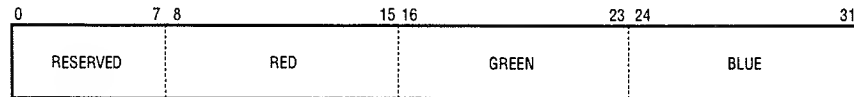


DBBMULTCONSTSSB1 - Source Coefficient Constant 1 Register (0x0004_806c)

This register contains the source RGB coefficient constant "1."

CLT_DBBMULTCONSTSSB1 (pp, red, green, blue)

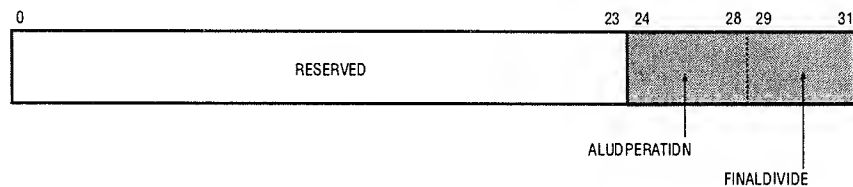
Field	Description
RED	Red constant 1
GREEN	Green constant 1
BLUE	Blue constant 1



DBALUCNTL - ALU Control Register (0x0004_8070)

CLT_DBALUCNTL (pp, aluoperation, finaldivide)

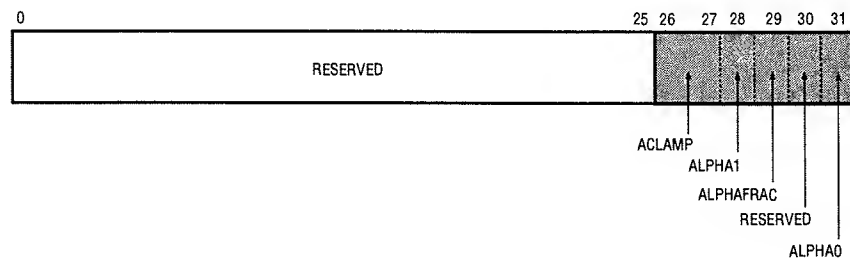
Field	Description
ALUOPERATION	ALU selection: A_PLUS_BCLAMP (0) A_PLUS_B (2) A_MINUS_BCLAMP (8) A_MINUS_B (a) B_MINUS_ACLAMP (c) B_MINUS_A (e) OUTPUTZERO (10) NEITHER (11) NOTA_AND_B (12) NOTA (13) NOTB_AND_A (14) NOTB (15) XOR (16) NOTA_AND_B (17) A_AND_B (18) ONEONEQUAL (19) B (1a) NOTA_OR_B (1b) A (1c) NOTB_OR_A (1d) A_OR_B (1e) OUTPUTONE (1f)
FINALDIVIDE	Final shift 0, 1, or 2



DBSRCALPHACNTL - Source Alpha Control Register (0x0004_8074)

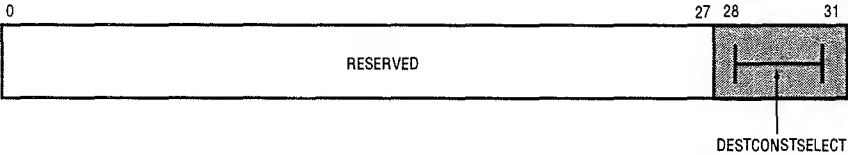
CLT_DBSRCALPHACNTL (pp, aclamp, alpha1, alphafrac, alpha0)

Field	Description
ACLAMP	Specify clamping option for three alpha ranges (3 @ 2 bit): LEAVEALONE (0) FORCE1 (1) FORCE0 (2)
ALPHA1	Two-bit field for Alpha equals 0xff
ALPHAFRAC	Two-bit field for fractional Alpha
ALPHA0	Two-bit field for Alpha equals 0

**DBDESTALPHACNTL - Destination Alpha Control Register (0x0004_8078)**

CLT_DBDESTALPHACNTL (pp, destconstselect)

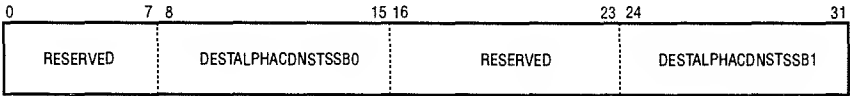
Field	Description
DESTCONSTSELECT	Combines meta select and select: TEXALPHA (0) TEXSSBCONST (1) SRCSSBCONST (9) SRCALPHA (2) RBLEND (3)



DBDESTALPHACONST - Destination Alpha Constants Register (0x0004_807c)

CLT_DBDESTALPHACONST (pp, destalphaconstssb0, destalphaconstssb1)

Field	Description
DESTALPHACONSTSSB0	Destination Alpha constant 0
DESTALPHACONSTSSB1	Destination Alpha constant 1



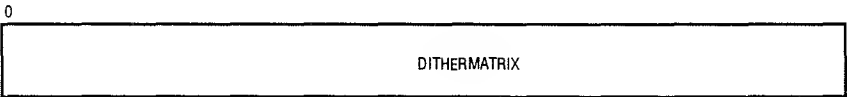
DBDITHERMATRIX - Dither Matrix Registers (0x0004_8080, 0x0004_8084)

This register contains the 4x4 dither matrix. The register at 0x0004_8080 holds the top two rows of the matrix, and is accessed using the *dmA* argument.

The register at 0x0004_8084 holds the bottom two rows of the matrix, and is accessed using the *dmB* argument.

CLT_DBDITHERMATRIX (pp, dmA, dmB)

Field	Description
X0Y0 - X3Y0/X0Y1 - X3Y1	First half of 4x4x4 dither matrix
X0Y2 - X3Y2/X0Y3 - X3Y3	Second half of 4x4x4 dither matrix

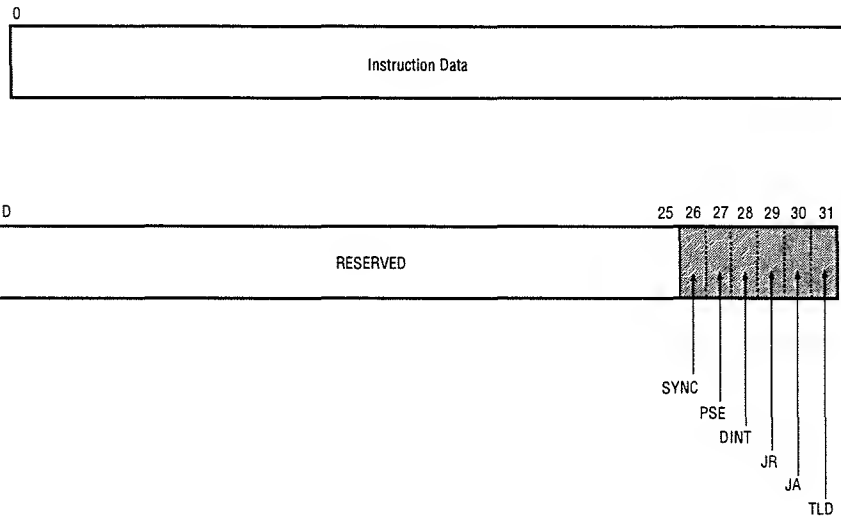


DCNTL - Deferred Flow Control Register (0x0004_0010, 0x0004_0014)

This register pair is used to perform various flow control instructions (see the "Flow Control Instructions" section, in Chapter 1, for more information). The data register is first loaded with the instruction data. The instruction register is then loaded with the instruction itself:

```
CLT_Sync (pp)
CLT_Pause (pp)
CLT_Interrupt (pp)
CLT_JumpRelative (pp)
CLT_JumpAbsolute (pp)
CLT_TxLoad (pp)
```

Field	Description
SYNC	Pause instruction execution until the entire TE pipe has been flushed
PSE	Pause execution
DINT	Interrupt CPU with vector - TEDCntlData is used as vector data
JR	Jump relative - use TEDCntlData as (2's complement) offset
JA	Jump absolute - use TEDCntlData as destination address
TLD	Texture load

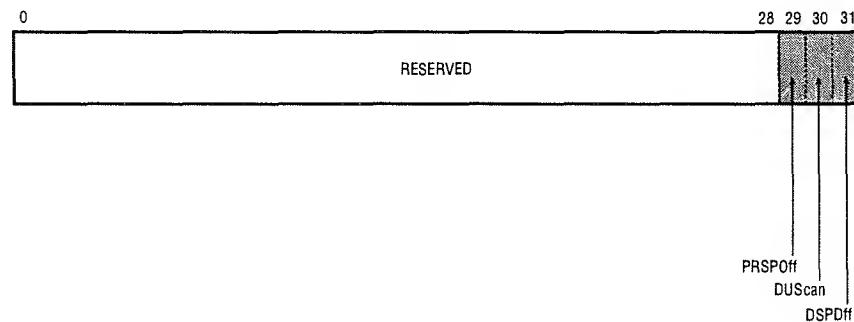


ESCNTL - Edge Walker/Span Walker Control Register (0x0004_4000)

This register contains three control bits. One turns perspective on and off, the second controls whether triangles are processed (scanned) from top-to-bottom or bottom-to-top, and the third turns DoubleStrike Prevention on and off.

CLA_ESCNTL (perspective, duscan, dspoff)

Field	Description
PRSPOff	Turn perspective calculations off.
DUScan	Down/up scan direction flags.
DSPOff	Disables Double Strike Prevention

**Register Memory Map**

Address	Name	Format	Access	Description
0x0004_8008	DBUserControl	Bit	RWSC	User Destination Blend Control
0x0004_800C	DBDiscard Control	Bit	RWSC	Pixel discard control
0x0004_8010	DBStatus	Bit	RWSC	Status indication
0x0004_8014	DBIntCntl	Bit	RWSC	Mask indicating which ALU and Z results should generate interrupts.
0x0004_801C	DBXWinClip	U_Fix	RW	X window clip values
0x0004_8020	DBYWinClip	U_Fix	RW	Y window clip values

Address	Name	Format	Access	Description
0x0004_8030	DBSrcCntl	Bit	RWSC	Source read Control register
0x0004_8034	DBSrcBaseAddr	Addr	RW	Source read base pointer
0x0004_8038	DBSrcXStride	U_Fix	RW	Source read X Stride value - Distance in pixels to next line.
0x0004_803C	DBSrcOffset	S_Fix	RW	X and Y Offset used for Source reads
0x0004_8040	DBZCntl	Bit	RWSC	Z Buffer Control register
0x0004_8048	DBZOffset	S_Fix	RW	X and Y Offset used for Z.
0x0004_8050	DBSSBDSBCntl	Bit	RWSC	SSB & DSB Control register
0x0004_8054	DBConstIn	U_Fix	RW	Const RGB used for computation input
0x0004_8058	DBAMultCntl	Bit	RWSC	Texture color multiplication control
0x0004_805C	DBAMultConst SSB0	U_Fix	RW	Texture RGB coef const "0"
0x0004_8060	DBAMultConst SSB1	U_Fix	RW	Texture RGB coef const "1"
0x0004_8064	DBBMultCntl	Bit	RWSC	Texture color multiplication control
0x0004_8068	DBBMultConst SSB0	U_Fix	RW	Source RGB consts "0"
0x0004_806C	DBBMultConst SSB1	U_Fix	RW	Source RGB consts "1"
0x0004_8070	DBALUCntl	Bit	RWSC	ALU Control register
0x0004_8074	DBSrcAlphaCntl	Bit	RWSC	Control of source alpha
0x0004_8078	DBDest AlphaCntl	Bit	RWSC	Control of destination alpha
0x0004_807C	DBDestAlpha-Const	U_Fix	RW	Destination Alpha constants

Address	Name	Format	Access	Description
0x0004_8080	DBDither MatrixA	Bit	RWSC	First half of 4x4x4 dither ma- trix
0x0004_8084	DBDither MatrixB	Bit	RWSC	Second half of 4x4x4 dither ma- trix
0x0004_808C- 0x0004_9FFC	reserved		NA	

Note: Registers are not initialized at reset unless stated otherwise. They must be explicitly set by the OS or user. Once a register has been set, it will retain its value throughout all rendering until it is altered by a list instruction or the CPU directly. Note that the triangle engine must go through a sync operation when a control register is changed. Changing registers "on the fly" will produce unpredictable results and in some circumstances may hang the triangle engine until it is reset.

Register Setups

This appendix lists the Triangle Engine register configurations for the following operations:

- ◆ Z-buffering
- ◆ Dithering
- ◆ Transparencies
- ◆ Texturing (on/off)
- ◆ Perspective (on/off)
- ◆ Tiling a texture
- ◆ Loading an uncompressed texture
- ◆ Loading a PIP table

Z-Buffering

Turn Z-buffering on:

```
CLT_SetRegister(pp,DBUSERCONTROL,CLA_DBUSERCONTROL(1,1,0,0,0,0,0,0))
```

Select which pixels are removed by the Z-buffer (only pixels that are closer than what is in the Z-buffer will be drawn):

```
CLT_DBZCNTL(ptr, 0,0,0,0,1,1)
```

Set the Z offset:

```
CLT_DBZOFFSET(ptr,0,0)
```

You will be unable to do Z-buffering unless you've allocated a Z-buffer and told the TE about it (typically with `GS_SetZBuffer`). You will also have to clear your Z-buffer every frame, and include perspective values (`w`) with your triangles.

Dithering

Turn dithering on:

```
CLT_SetRegister(pp,DBUSERCONTROL,CLA_DBUSERCONTROL(0,0,0,0,0,0,1,0))
```

Set the dithering matrix to a basic dither pattern (see Figure A-1):

```
CLT_DbDitherMatrix(ptr,0xC0D12E3F,0xE1D0302F)
```

$$\begin{bmatrix} -4 & 0 & -3 & 1 \\ 2 & -2 & 3 & -1 \\ -2 & 1 & -3 & 0 \\ 3 & 0 & 2 & -1 \end{bmatrix}$$

Figure E-1 *Dithering matrix*

Setting Up the Destination Blender Source Buffer

To set up the frame buffer as a secondary source, set the following registers:

DBUSERCONTROL can be used to turn blending on, and to enable blending with a source frame buffer.

DBSRCBASEADDR points to the address of the frame buffer bitmap

DBSRCCNTL specifies whether the frame buffer is 16 or 32 bits

DBSRCOFFSET specifies an X and Y offset to use before reading from the frame buffer. This should usually be set to (0,0)

DBSRCXSTRIDE sets the width of the source frame buffer

Transparencies

There are a number of ways to do a transparency:

- To cause black (RGB 0,0,0) pixels, pixels with SSB 0, or pixels with Alpha 0 to be totally transparent, simply use the DBDISCARDCONTROL register. (Note: this checking takes place after all texturing has been done.)
- To set a constant amount of transparency (for instance, 90%), first set up the frame buffer as a secondary source (see above). Then call:

```
CLT_DBAMULTCNTL(ptr,  
RC_DBAMULTCNTL_AINPUTSELECT_TEXCOLOR,  
RC_DBAMULTCNTL_AMULTCOEFSELECT_CONST, 0, 0) to set the first source  
to come from the triangle data
```

```
CLT_DBBMULTCNTL(ptr, RC_DBBMULTCNTL_BINPUTSELECT_SRCCOLOR,  
RC_DBBMULTCNTL_BMULTCOEFSELECT_CONST, 0, 0) to set the second  
source to come from the source data
```

```
CLT_DBALUCNTL(ptr, RC_DBALUCNTL_ALUOPERATION_A_PLUS_B, 0) to  
set the final math to add the two sources together
```

```
CLT_DBAMULTCONSTSSB0(ptr, 255-TRANS, 255-TRANS, 255-TRANS)
```

```
CLT_DBAMULTCONSTSSB1(ptr, 255-TRANS, 255-TRANS, 255-TRANS)
```

CLT_DBBMULTCONSTSSB0(ptr, TRANS, TRANS, TRANS)

CLT_DBBMULTCONSTSSB1(ptr, TRANS, TRANS, TRANS) to set up the degree of transparency, where TRANS is between 0 and 255, with 0 meaning totally opaque and 255 meaning totally transparent

- To set transparency to depend on the Alpha channel, first set up the frame buffer as a secondary source (see above). Then call:

```
CLT_DBAMULTCNTL(ptr,
RC_DBAMULTCNTL_AINPUTSELECT_TEXCOLOR,
RC_DBAMULTCNTL_AMULTCOEFSELECT_TEXALPHA, 0, 0);
```

```
CLT_DBBMULTCNTL(ptr, RC_DBBMULTCNTL_BINPUTSELECT_SRCCOLOR,
RC_DBBMULTCNTL_BMULTCOEFSELECT_TEXALPHACOMPLEMENT, 0, 0);
```

```
CLT_DBALUCNTL(ptr, RC_DBALUCNTL_ALUOPERATION_A_PLUS_B, 0)
```

Then an alpha value of 1.0 will result in a completely opaque triangle, while an alpha of 0.0 will result in a completely transparent triangle, etc.

Texturing

To turn texturing on and off, use the lookup enable bit of the TXTADDRCNTL register. If you leave this bit set, you may get a performance hit even if you are drawing untextured triangles. However, there are a number of other registers that must be set before texturing can be used (not to mention that a texture load must be done before a texture can be drawn):

TXTADDRCNTL must also be used to select a filtering mode and set the number of LODs present.

TXTPIPCNTL must be set, even if you're using a literal (unindexed) texture. For a literal texture, it should be set so that the SSB, alpha and color come from the texture cache. For an indexed texture, the color should presumably come from the PIP, and the alpha and SSB can come from wherever you like. For indexed textures, this register also contains the PIP index offset, allowing you to select which part of the PIP will be used.

TXTEXPTYPE must be set. This specifies what type of texture is being used (how many bits of color, whether it is indexed or unindexed, etc.) This will typically be set during your texture load

TXTUVMASK must be set. This register is used to tile textures (see below), but for untiled textures it should be set to (0x3ff, 0x3ff)

TXTUVMAX must be set. This sets the size of the texture, and will typically be set during your texture load. If multiple LODs are present, this register refers to the size of the smallest, and thus coarsest, LOD. Note that the values contained in this register are the dimensions of the texture, in pixels, minus one.

TXTLODBASE[0-(n-1)] must be set, where n is the number of LODs present. These registers specify the starting locations of the texture LODs in TRAM. They will usually be set during the texture load.

TXTTABCNTL must be set up properly. If texturing is turned off, then there will be only two inputs to the texture blender (PRIMCOLOR and PRIMALPHA) so any blends used can only have them as inputs. Typically, you will just want to set COLOROUT to PRIMCOLOR and ALPHAOUT to PRIMALPHA. If texturing is

turned on, then there are many many more options. However, if you actually want your texture to show up, you'll need to set COLOROUT to either TEXCOLOR or some BLEND one of whose inputs is TEXCOLOR.

ESCNTL must be used to turn perspective correction either on or off, according to your needs. If it is set incorrectly, your textures will be warped and unviewable.

Perspective

Turn perspective correction on:

```
CLT_SetRegister(pp, ESCNTL, CLA_ESCNTL (1,0,0))
```

Turn perspective correction off:

```
CLT_ClearRegister(pp, ESCNTL, CLA_ESCNTL (1,0,0))
```

Remember that if perspective correction is on, the texture coordinates entered in vertex commands must be multiplied by $1/w$, but if perspective correction is off, raw texture coordinates must be used.

Sprites

To draw sprites, that is, to use the TE to simply draw textures onto the screen very quickly, make sure that blending is off, secondary source is off, perspective correction is off, Z-buffering is off (unless it's needed), and point filtering is being used. Be aware, however, that if you just want to copy large rectangular regions of data with the TE, it's faster to set a secondary source to point to the source data, enable source blending in the destination blender, and use a strip or fan of two untextured triangles to copy the data. However, this will only work if the source data is in literal 16- or 32-bit format, and if no special effects (such as black regions being transparent) are needed.

Lighting

To do lighting, you'll first need a 3D lighting model, which is a very complicated issue in its own right. However, there are a few typical things that might be done with the CLT to implement the graphical side of the lighting model. In particular, a lighting model will frequently be used with textured triangles, and it will want to do two things, namely darken the texture where the texture is not illuminated, and brighten the texture where there should be specular highlights.

To do either of these things on a triangle-by-triangle basis requires that there be color data for the vertices, but that color data can be used several different ways.

Lighting method #1: With this method, the RGB values of the color data are used to select the color of a light, and the alpha value sets how strong that light is. More specifically, the texture application blender is used, with the iterated alpha controlling the LERP between the iterated color and the textured color. So, to darken the texture, you can specify an RGB color of black. Then an alpha of 1 would result in black, an alpha of 0.5 would result in the texture being displayed at 50% intensity, an alpha of 0 would result in the normal texture, and so on. To brighten the texture, you can specify on RGB of white, or whatever color you want your specular highlights to be. The advantage of this method is that it uses only the texture application blender, so the destination blender is free to produce other effects, or to be turned off to save performance. Furthermore, this method allows you to still use your textured alpha for transparency or whatever, even though your iterated alpha is being taken up in the lighting. The disadvantage is

that it assumes that you've combined all of your lighting calculations and arrived at one final output color, instead of keeping the darkening and brightening effects separate. To use this method, simply set up the texture application blending as follows:

```
CLT_TXTTABCNTL(ptr, RC_TXTTABCNTL_FIRSTCOLOR_TEXCOLOR,
RC_TXTTABCNTL_SECONDCOLOR_PRIMCOLOR,
RC_TXTTABCNTL_THIRDCOLOR_PRIMALPHA, 0, 0,
RC_TXTTABCNTL_COLOROUT_BLEND,
RC_TXTTABCNTL_ALPHAOUT_TEXALPHA,
RC_TXTTABCNTL_BLENDOP_LERP);
```

Lighting method #2: With this method, the RGB color is used to darken the texture, and the alpha value is used to brighten the color. In this case, both the texture and destination blenders are used, with the darkening taking place in the texture blender and the lightening taking place in the destination blender. With this method, the RGB values are used as multipliers for the texture colors, so RGB values of (0.5, 0.5, 0.5) will result in the texture being displayed at half intensity. However, the alpha value is used to brighten the color, after the darkening has taken place, with an alpha of 0 not affecting the texture at all, and an alpha of 1 brightening all the way to solid white. This method is easier to use with most lighting models that treat specular highlights separately from the other lighting, but it uses both the texture and destination blenders, and doesn't allow the texture alpha to be accessed at all. To use this method, set up the blending as follows:

```
CLT_TXTTABCNTL(ptr, RC_TXTTABCNTL_FIRSTCOLOR_TEXCOLOR,
RC_TXTTABCNTL_SECONDCOLOR_PRIMCOLOR,
RC_TXTTABCNTL_THIRDCOLOR_PRIMALPHA, 0, 0,
RC_TXTTABCNTL_COLOROUT_BLEND,
RC_TXTTABCNTL_ALPHAOUT_PRIMALPHA,
RC_TXTTABCNTL_BLENDOP_MULT);

CLT_DBAMULTCNTL(ptr,
RC_DBAMULTCNTL_AINPUTSELECT_TEXCOLOR,
RC_DBAMULTCNTL_AMULTCOEFSELECT_CONST, 0, 0);

CLT_DBAMULTCONSTSSB0(GS_Ptr(gs), 255, 255, 255);
CLT_DBAMULTCONSTSSB1(GS_Ptr(gs), 255, 255, 255);

CLT_DBBMULTCNTL(ptr,
RC_DBBMULTCNTL_BINPUTSELECT_TEXCOLORCOMPLEMENT,
RC_DBBMULTCNTL_BMULTCOEFSELECT_TEXALPHA, 0, 0);

CLT_DBALUCNTL(ptr, RC_DBALUCNTL_ALUOPERATION_A_PLUS_B, 0)
```

Simpler lighting models: If your lighting model doesn't require specular highlights, it is easy to simply have the RGB color or the alpha value multiply the texture value. This is accomplished with the texture application blender. For instance, to have the alpha value multiply the texture value, call:

```
CLT_TXTTABCNTL(ptr, RC_TXTTABCNTL_FIRSTCOLOR_TEXCOLOR,
RC_TXTTABCNTL_SECONDCOLOR_PRIMALPHA, 0, 0, 0,
RC_TXTTABCNTL_COLOROUT_BLEND,
RC_TXTTABCNTL_ALPHAOUT_TEXALPHA,
RC_TXTTABCNTL_BLENDOP_MULT);
```

Tiling a Texture

First of all, you can only tile a texture if its width or height (depending on which direction you're tiling in) is a power of two. To turn on tiling, you'll need to set the `TEXTUVMASK` and `TEXTUVMAX` registers in a fairly trick fashion. Each of these registers has both an X- and a Y-component. To tile your texture in the X direction, first set the X component of `TEXTUVMASK` to `txwidthmax-1`, where `txwidthmax` is equal to the width, in pixels, of the largest LOD. Then set the X component of `TEXTUVMAX` to `0x3ff >> (numLOD-1)`, where `numLOD` is the number of LODs present in the texture. To turn off tiling in the X direction, set the X component of `TEXTUVMASK` to `0x3ff` and the X component of `TEXTUVMAX` to `txwidth-1`, where `txwidth` is the width, in pixels, of the smallest (coarsest) LOD.

Similarly, to tile your texture in the Y direction, set the Y component of `TEXTUVMASK` to `txheightmax-1` and the Y component of `TEXTUVMAX` to `0x3ff >> (numLOD-1)`. To turn tiling off in the Y direction, set the Y component of `TEXTUVMASK` to `0x3ff` and the Y component of `TEXTUVMAX` to `txheight-1`.

Load an Uncompressed Texture

To do a texture load of an uncompressed texture, make the following calls:

```
CLT_TXTLDCNTL(ptr, 0, RC_TXTLDCNTL_LOADMODE_TEXTURE, bitoffset)
```

Where `bitoffset` is the offset into the byte in main memory of the start of the texture to be loaded, in bits.

```
CLT_TXTCOUNT(ptr, textureheight)
```

Where `textureheight` is the number of rows in the texture.

```
CLT_TXTLDSRCADDR(ptr, txaddress)
```

Where `txaddress` is the address in memory of the texture to be loaded.

```
CLT_TXTLODBASE0(ptr, tramaddress)
```

Where `tramaddress` is the word-aligned address in TRAM where the texture will be placed (ranging from 0 to 0x3fff). If you're only going to have one texture in TRAM at a time, this should probably be 0.

```
CLT_TXTLDWIDTH(ptr, txwidth, txsourcewidth)
```

Where `txwidth` is the width of the texture, in bits, and `txsourcewidth` is the width of the region from which the texture is being loaded, in bits.

```
CLT_TxLoad(ptr)
```

Load the texture.

Loading a PIP Table

To perform a PIP load, make the following calls:

```
CLT_TXTLODBASE0(ptr, 0x46000)
```

Tells the texture loader to load the PIP into PIP RAM.

```
CLT_TXTLDCNTL(ptr, 0, RC_TXTLDCNTL_LOADMODE_PIP, 0)
```

```
CLT_TXTLDSRCADDR(ptr, pipsourceaddress)
```

Where `pipsourceaddress` is the word-aligned address in main memory where the PIP is located.

CLT_TXTCOUNT(ptr, pipsize)

Where `pipsize` is the size, in bytes, of the PIP to load. A full 256 entry PIP takes up 1024 bytes.

CLT_TxLoad(ptr)

Sample Code

This is a sample program that uses Gstate and the Command List Toolkit. It draws random triangles up on the screen very rapidly.

```
/*All of the following are nice basic include files*/
#include <stdio.h>
#include <stdlib.h>
#include <kernel:types.h>
#include <graphics:clt:clt.h>
#include <graphics:clt:gstate.h>
#include <graphics:graphics.h>
#include <graphics:view.h>
#include <graphics:error.h>
#include <kernel:io.h>
#include <kernel:mem.h>
#include <kernel:random.h>
#include <assert.h>

#define TRISPERLIST 100

/*This constant specifies how many triangles each command list will draw. If
 *we wanted to we could have each command list we send to the TE have only
 *one triangle in it, but there is some overhead to sending lists, so it's
 *best to have each list not be tiny.
 */

int32 main(void)
{
    GState *gs;
    /*the heart of using GState is creating a GState struct... */

    Item viewItem; /*we need a View so that things will show up on screen*/
```

```
int i;

Item bitmaps[1];
/*we need a bitmap item for the view to display,
 *and for the TE to render into
 */

OpenGraphicsFolio (); /*must always be called before doing graphics...*/

gs=GS_Create();
/*This call creates and initializes the GState*/

GS_AllocLists(gs, 2, 4096);
/*This call allocates the command lists for the GState. In this case,
 *we'll have two command lists, each of 4096 words
 */

GS_AllocBitmaps( bitmaps, 320, 240, BMTYPE_l6, 1, 0);
/*this call allocates a bitmap for us*/

GS_SetDestBuffer(gs, bitmaps[0]);
/*this call tells the GState which bitmap to render into*/

viewItem = CreateItemVA(MKNODEID(NST_GRAPHICS, GFX_VIEW_NODE),
    VIEWTAG_VIEWTYPE, BMTYPE_l6,
    /*specifies 320x240x16bit*/

    VIEWTAG_BITMAP, bitmaps[0],
    /*Point the view towards our first bitmap*/

    TAG_END );
/*this call creates the View Item, which is necessary for
 *things to show up on screen
 */

AddViewToViewList( viewItem, 0 );
/*This causes the view to actually be displayed*/

/*the next several commands are all CLT commands, all of
 *which are register commands. For instance, the first
 *command is CLT_DBUSERCONTROL. It's passed a pointer to
 *a command list, and a bunch of numbers. What it does is
 *it adds a command to the command list. That command
 *sets the state of the DBUSERCONTROL register, which is
 *nice, basic TE register. The value it sets DBUSERCONTROL
 *to comes from the arguments. In the case of DBUSERCONTROL,
 *most of these arguments represent a single flag bit.
 */

/*note that the moral of this story is that before _anything_
 *will show up on the screen, there are quite a few registers
 *that need to be set up, but once they're all set up, they
 *can be more or less ignored
 */
```

```

CLT_DBUSERCONTROL(GS_Ptr(gs), 0,0,0,0,1,1,0,15);
/*basic initialization. In this case, Z-buffering is off,
 *window clipping is off, destination blending is on, source
 *blending is on, and dithering is off
 */

/*this next command also is a register setting command,
 *in this case setting the DBDISCARDCONTROL register
 */
CLT_DBDISCARDCONTROL(GS_Ptr(gs),0,0,0,0);
/*all pixel discards are turned off. these would be turned on, for
 *instance, to make all black pixels transparent
 */

CLT_DBCONSTIN(GS_Ptr(gs), 0,0,0);
/*set the constant color used in destination blending to black
 *(destination blending takes the color and texture from the triangle
 *and does a number of odd things to it. In this case, we'll just add
 *it to black, which effectively leaves it unchanged)
 */

CLT_DBAMULTCONSTSSB0(GS_Ptr(gs), 0xff, 0xff, 0xff);
CLT_DBAMULTCONSTSSB1(GS_Ptr(gs), 0xff, 0xff, 0xff);
CLT_DBBMULTCONSTSSB0(GS_Ptr(gs), 0xff, 0xff, 0xff);
CLT_DBBMULTCONSTSSB1(GS_Ptr(gs), 0xff, 0xff, 0xff);
/*set all of our multiplying constants for the destination blender
 *to 1, so we can just pass colors directly through
 */

CLT_DBALUCNTL(GS_Ptr(gs), RC_DBALUCNTL_ALUOPERATION_A_PLUS_B, 0);

/*This set the final ALU stage of the destination blending to just add the
 *A and B colors together. In this case, the A color will be the
 *shading from the triangle we're drawing and the B color will be black
 */

CLT_TXTADDRCNTL(GS_Ptr(gs), 0, 0, 0, 0, 0);
/*Turn texturing off*/

CLT_DBAMULTCNTL(GS_Ptr(gs), RC_DBAMULTCNTL_AINPUTSELECT_TEXCOLOR,
RC_DBAMULTCNTL_AMULTCOEFSELECT_CONST, RC_DBAMULTCNTL_AMULTCONSTCONTROL_TEXSSB,
0);

/*sets the A input to the destination blender to be the color from the texture
 *mapper. Note that even though we aren't doing any texturing we want the color
 *from the texture mapper, because the texture mapper handles all shading.
 */

CLT_DBBMULTCNTL(GS_Ptr(gs), RC_DBBMULTCNTL_BINPUTSELECT_CONSTCOLOR,
RC_DBBMULTCNTL_BMULTCOEFSELECT_CONST,
RC_DBBMULTCNTL_BMULTCONSTCONTROL_TEXSSB, 0);

/*sets the B input to the destination blender to just be a constant color

```

```
    *(which in this case we earlier set to black)
    */

    GS_SendList(gs);

/*This command is totally different from all the CLT macros we've been using.
*It tells the GState to take the command list it's been assembling, which
*consists of a bunch of register instructions, and send it to the TE.
*Note that whenever you call SendList the Gstate will stick a CLT_Pause
*command at the end of your command list, which means that you don't have
*to remember to do so yourself. But if you're ever sending lists
*yourself, always remember to terminate them with a CLT_Pause command.
*/

while(1)
{
    for (i=0;i<TRISPERLIST;i++)
    {

        CLT_TRIANGLE(GS_Ptr(gs), 1, RC_STRIP, 0, 0, 1, 3);
        /*To set up a command list to draw a triangle, we first need a
        *triangle command. The second argument says that this is a new
        *tristrip/fan (as opposed to connecting to a previously drawn one.
        *The third argument says that it's a strip, not a fan.
        *The fourth argument says that it has no perspective info (z
        *values). The fifth argument says that it has no texturing.
        *The sixth argument says that it does have RGBA color info.
        *The final argument says that it has 3 vertices.
        *
        *After the triangle command we'll need to issue one vertex command
        * for each vertex of the triangle. Note that there can not be any
        *other TE instructions between the triangle command and the vertex
        *commands. Also, if you give the wrong number of vertex commands,
        *or if you, say, include
        *perspective info in the vertex command but don't mention it in the
        *triangle command, then bad things will happen. Namely, the TE will
        *probably time out.
        */

        CLT_VertexRgba(GS_Ptr(gs), (gfloat)(rand()%320), (gfloat)(rand()%240),
            (gfloat)(rand() % 10000)/10000.0,
            (gfloat)(rand() % 10000)/10000.0, (gfloat)(rand() % 10000)/10000.0,0);
        CLT_VertexRgba(GS_Ptr(gs), (gfloat)(rand()%320), (gfloat)(rand()%240),
            (gfloat)(rand() % 10000)/10000.0,
            (gfloat)(rand() % 10000)/10000.0, (gfloat)(rand() % 10000)/10000.0,0);
        CLT_VertexRgba(GS_Ptr(gs), (gfloat)(rand()%320), (gfloat)(rand()%240),
            (gfloat)(rand() % 10000)/10000.0,
            (gfloat)(rand() % 10000)/10000.0, (gfloat)(rand() % 10000)/10000.0,0);

        /*CLT_VertexRgba is the macro for vertices with shading but no perspective
        *or texture. Since perspective, texture, and shading can all be turned
        *on or off independently, there are a total of 8 possible distinct
        *types of vertex command, each with an accompanying macro.
        */
    }
}
```



```
        CLT_Sync(GS_Ptr(gs));

        /*after drawing a triangle, but before setting any registers, you should
        *always call CLT_Sync, which inserts a sync command into the command list.
        *In this particular case, we don't set any registers after drawing the
        *triangle, so we don't really need this sync command, but I needed an
        *excuse to mention it, since it's so important. Get in the habit of
        *sticking it after triangle commands.
        */
    }

    GS_SendList(gs);
    /*This sends our newly created command list with instructions for
    * drawing 100 triangles to the Triangle Engine
    */

    }
    CloseGraphicsFolio();
    /*and we're done!*/

    }
    /*have a nice day*/
```


Function Calls

This Appendix lists all of the Command List Toolkit function calls.

Index

A

- aliasing
 - avoiding (with mipmaps) CLT-21
- AllocSignal() CLT-14
- alpha component (of a texel) CLT-28
- alpha value
 - dynamic range of CLT-34
- alpha value, see also alpha component CLT-28

B

- Bi-linear filtering CLT-21
- bi-linear filtering CLT-26

C

- color
 - of 3D objects CLT-17
 - primitive CLT-34
- color component (of a texel) CLT-28
- color components
 - of a texel CLT-32
- color constants CLT-34
- color table CLT-33
- color value, see also color component CLT-28
- command list buffers CLT-10, CLT-11, CLT-13
 - space CLT-13
- command lists CLT-1
- components
 - color (of a texel) CLT-32
 - of a texel CLT-28

- compression
 - data CLT-34
 - of texels CLT-28
- constant registers CLT-28
- constants
 - color CLT-34

D

- data compression CLT-34
- DBALUCNTL CLT-73
- DBAMULTCNTL CLT-69
- DBAMULTCONSTSSB0 CLT-70
- DBAMULTCONSTSSB1 CLT-71
- DBBMULTCNTL CLT-71
- DBBMULTCONSTSSB0 CLT-72
- DBBMULTCONSTSSB1 CLT-72
- DBCONSTIN CLT-69
- DBDESTALPHACNTL CLT-74
- DBDESTALPHACONST CLT-75
- DBDISCARDCONTROL CLT-64
- DBDITHERMATRIX CLT-75
- DBINTCNTL CLT-64
- DblARightJustify attribute CLT-61
- DblBRightJustify attribute CLT-61
- DblEnableAttrs attributes CLT-63
- DblFinalDivide attribute CLT-61
- DblXWinClipMax CLT-63
- DblXWinClipMin CLT-63
- DblYWinClipMax CLT-63
- DblYWinClipMin CLT-63
- DBSRCALPHACNTL CLT-74
- DBSRCBASEADDR CLT-66
- DBSRCCNTL CLT-66

DBSRCOFFSET CLT-67
DBSRCXSTRIDE CLT-66
DBSSBDSBCNTL CLT-68
DBXWINCLIP CLT-65
DBYWINCLIP CLT-65
DBZCNTRL CLT-67
DBZOFFSET CLT-68
DCNTL CLT-76
distortion

 and textures CLT-19
double-buffering CLT-14

E

ESCNTL CLT-77

F

filtering

 bi-linear, see also bi-linear filtering CLT-26
 in texture-mapping CLT-19
 linear, see also linear filtering CLT-24
 mipmap CLT-21
 nearest (point), see also nearest (point) filtering CLT-24
 quasi tri-linear, see also quasi tri-linear filtering CLT-26

filtering modes

 mipmap CLT-21

flow control instructions

 INT CLT-9
 JA CLT-9
 JR CLT-9
 PAUSE CLT-9
 SYNC CLT-9
 TXLD CLT-9

frame buffer bitmaps CLT-13

G

ghost

 rendering CLT-33

GS CLT-15

GS_BeginFrame() CLT-14

GS_Create() CLT-14

GS_Delete() CLT-15

GS_FreeBuffers() CLT-15

GS_GetCurListStart() CLT-14

gs_ListPtr field CLT-13

GS_Reserve() CLT-13

GS_Reserve() CLT-13

gs_SendList CLT-13

GS_SetList() CLT-14

GS_SetVidSignal() CLT-14

GS_WaitIO CLT-14

GS_WaitIO() CLT-14

I

interfilter CLT-25, CLT-26

Introduction CLT-2

L

level CLT-20

level of detail (LOD) CLT-20

level of detail, see also LOD CLT-20

Linear filtering CLT-21

linear filtering CLT-24
 equation for CLT-25

M

magnification filter CLT-25

mapping

 PIP CLT-32, CLT-33

 texture CLT-19

minification filter CLT-25

mipmap

 described CLT-20

 sizes of CLT-20

mipmap filtering CLT-21

mipmap filtering modes CLT-21

mipmap texture filtering CLT-19

mipmapping CLT-19

mipmaps CLT-19

modes

 filtering (for mipmaps) CLT-21

ModifyGraphicsItem() CLT-15

multim im parvo CLT-20

N

Nearest (point) filtering CLT-21

nearest (point) filtering CLT-24, CLT-26

O

object
 textured CLT-19
offset
 PIP CLT-33

P

palette index table, see also PIP table CLT-32
palette lookup table, see also PIP table CLT-32
Pen Index Palette CLT-4
PIP mapping CLT-32, CLT-33
PIP mapping stage CLT-33
PIP offset CLT-33
PIP table CLT-32
PIP-table entry CLT-32
pixel
 perspective correction CLT-6
 texture coordinates CLT-6
 x, y coordinates CLT-6
pixel scaling CLT-61
primitive color CLT-34

Q

Quasi tri-linear filtering CLT-21
quasi tri-linear filtering CLT-26

R

red, green, blue, and alpha components CLT-6
Register Commands CLT-63
registers
 constant CLT-28
rendering a ghost CLT-33

S

scaling
 pixel, see pixel scaling
shift CLT-7
source selection bit, see also SSB CLT-28

space CLT-13
SSB CLT-28, CLT-33
SYNC CLT-4, CLT-9

T

table
 color CLT-33
 PIP, see also PIP table CLT-32
TE, see also Triangle Engine CLT-61
tear-free rendering CLT-13
texel CLT-18
 alpha component CLT-28
 blending CLT-34
 color component CLT-28
 components of CLT-28, CLT-34
 compression of CLT-28
 defined CLT-18
 interpreting color data in CLT-33
 mapping to pixels CLT-19
texture
 address of CLT-18
 coordinate system of CLT-18
 defined CLT-17
texture application blending CLT-36
texture filtering CLT-35
 example of CLT-24
 mipmap CLT-19
texture mapping CLT-19
texture mapping step by step CLT-35
texture RAM, see also TRAM CLT-28
texture-mapping
 example of CLT-33
Textures
 tiled CLT-6
textures
 storing CLT-18
TRAM CLT-33
TRAM (texture RAM) CLT-28
transparency
 of 3D objects CLT-17

Triangle Engine

color calculations performed by CLT-61

deferred mode CLT-4

Destination Blender CLT-2

state registers

shifts and masks CLT-7

Texture Mapper CLT-2

tri-linear filtering

quasi, see also quasi tri-linear filtering CLT-26

U

u coordinate CLT-18

uint32 CLT-34

V

v coordinate CLT-18

vertex header instruction CLT-5

W

WinClipInEnable CLT-63

WinClipOutEnable CLT-63



3DO M2 Video Processing Guide

Version 2.0 – May 1996

Copyright © 1996 The 3DO Company and its licensors.

All rights reserved. This material constitutes confidential and proprietary information of The 3DO Company. This documentation is subject to a license agreement with The 3DO Company and may be used only by parties to such agreement. Use by any other persons, and/or for any purpose not expressly authorized by the agreement, is strictly prohibited.

3DO's LICENSOR(S) MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. 3DO'S LICENSOR(S) DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME JURISDICTIONS. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

Other brand or product names are the trademarks or registered trademarks of their respective holders.

Contents

Preface

About this document	VID-ix
Audience.....	VID-ix
How this document is organized	VID-ix
Typographical conventions.....	VID-x

1

Overview

Introduction.....	VID-1
Chapter Overview	VID-2
The Main Paths Available.....	VID-2
What You Start With	VID-2
The Overall Process	VID-2
The Main Paths You can Take - MPEG or EZFlix (and DSP Audio)	VID-3
Factors in Choosing a Path	VID-3
Diagram of Video Processing Paths	VID-5
The MPEG Path	VID-5
Overview	VID-5
Hardware MPEG Encoding With No Editing	VID-6
Hardware MPEG Encoding With Editing	VID-7
Software MPEG Encoding	VID-8
The EZFlix Path.....	VID-10
The DSP Audio Path	VID-11

2 Digitizing

Introduction	VID-13
Source Material	VID-14
Shooting Your Own Video	VID-14
Film-Based Source	VID-14
Digitizing Video.....	VID-15
Digitizing Process	VID-16
Processing Digitized Materials.....	VID-17
3-2 Pulldown	VID-18
Deinterlacing	VID-18
Cropping and Resizing	VID-18

3 Compression

Introduction.....	VID-21
Which Compression Algorithm.....	VID-21
EZFlx	VID-22
Cinepak (Not Currently Supported in 3DO M2)	VID-23
MPEG	VID-23
QuickTime and Cinepak Tools.....	VID-24
Size Constraints	VID-24
EZFlx Advantages	VID-24
Setting Data Rate	VID-25
Compression Options.....	VID-25

4 MPEG Encoding with Sparkle

Introduction.....	VID-27
Overview	VID-27
Related Documentation	VID-28
About Sparkle	VID-28
Requirements	VID-28
Preprocessing.....	VID-28
Some Considerations.....	VID-29

Encoding with Sparkle	VID-30
Setting MPEG Playback Preferences	VID-30
Setting MPEG Encoding Parameters	VID-32
Fine Tuning	VID-33
Putting MPEG into a Stream	VID-35

5

Video Tools

Introduction	VID-37
The 3-2 Pulldown Tool	VID-37
Understanding Telecine	VID-38
Choosing a Movie	VID-39
Field Dominance	VID-39
Removing Composite Frames	VID-40
Saving a Processed Movie	VID-42
Troubleshooting	VID-42
MovieEdit	VID-43
Saving Movies with MovieEdit	VID-43
MovieEdit User Interface	VID-44
Limitations	VID-45
Tips and Tricks	VID-45
MovieCompress	VID-45
Movie Compression Settings Dialog Options	VID-45
Using the MovieCompress Tool	VID-46
EZFluxChunkifier	VID-48
EZFluxChunkifier Command-Line Arguments	VID-48
Weaving	VID-49

6

Compressing with EZFlux

Introduction	VID-51
About this Chapter	VID-51
Audience	VID-52
EZFlux Functionality	VID-52
Caveats	VID-53
Installation	VID-53
Compressing a Movie	VID-53
The Weave Script	VID-53

Playing a Movie	VID-54
Player Controls	VID-55
Index	VID-57

List of Figures

- Figure 1-1: Overview of video processing paths VID-5
- Figure 1-2: Hardware MPEG encoding with no editing VID-6
- Figure 1-3: Hardware MPEG encoding with editing VID-7
- Figure 1-4: Software MPEG encoding using Sparkle VID-9
- Figure 1-5: EZFlix processing VID-10
- Figure 1-6: DSP audio processing VID-11
- Figure 4-1: MPEG Playback Preferences. VID-30
- Figure 4-2: MPEG Encoding dialog. VID-32
- Figure 4-3: Compressing 44K-sampled 16-bit stereo sound VID-35
- Figure 5-1: The Telecine process inserts frames. VID-38
- Figure 5-2: Composite frames show misaligned fields. VID-40
- Figure 5-3: Correctly and incorrectly marked composite frames. VID-41
- Figure 5-4: MovieEdit window. VID-43
- Figure 5-5: Compression Settings dialog VID-47

Preface

About this document

The Video Processing Guide describes how to prepare and process video materials into digital video clips for 3DO™ title development, discusses hardware options, and suggests a number of software solutions.

Audience

This document is for 3DO developers—including programmers, video specialists, graphic artists, and project managers—using digital video in title development.

How this document is organized

- ◆ Chapter 1, “Overview,” defines the basic paths to video processing currently available in the 3DO development environment, and follows each of these paths from the capture phase through the editing and compression phases, up to exporting the movie to the 3DO file format.
- ◆ Chapter 2, “Digitizing,” explains how to prepare materials for capture, how to capture them, and how to process them in preparation for compression.
- ◆ Chapter 3, “Compression,” begins with a comparative analysis of EZFlix, Cinepak (referred to as Compact Video in Macintosh documentation), and MPEG video compression, and then looks at some quality issues.
- ◆ Chapter 4, “MPEG Encoding with Sparkle” discusses encoding for the 3DO environment with Sparkle, a freeware program that encodes QuickTime source into an MPEG Video Elementary Stream.
- ◆ Chapter 5, “Video Tools,” covers the video processing tools provided or supported by 3DO. These include:
 - ◆ The 3-2 Pulldown tool for processing film-based video.

- ◆ MovieEdit, which lets you play and edit QuickTime movies. Dynamic displays of time code, time unit, and frame number information give users greater flexibility and precision in editing movies.
- ◆ MovieCompress for compressing movies using Apple QuickTime codecs. MovieCompress lets you specify frame rate, key frame rate, data rate, quality, depth, and codec. It also offers a drag-and-drop feature for opening a movie file, and is AppleScript-able for the four standard events (Open, OpenDoc, Quit, Save).
- ◆ EZFlixChunkifier, an MPW Tool that does EZFlix compression and chunkification in a single step.
- ◆ Chapter 6, "Compressing with EZFlix" discusses EZFlix, a software 3DO video compression tool and playback engine for creating and playing back compressed digital video.

Typographical conventions

The following typographical conventions are used in this document:

Item	Example
code example	<code>int32 OpenGraphicsFolio(void)</code>
procedure name	CreateScreenGroup
new term or emphasis	That added weight is called a <i>cornerweight</i> .
file or folder name	Open the <i>red cd</i> file in the <i>cdrommaster</i> folder.

Overview

Introduction

Video production in 3DO title development is a process complicated not only by the several processing paths currently available, but by a growing number of video processing tools that add forks to the main paths, and options that fork these alternate paths almost ad infinitum. Add to this the dizzying array of hardware requirements, from pre-production components to audio and video cards and cables, and even the veteran video producer needs a compass to navigate this tangle of criss-crossing paths.

The process is further complicated when the audio track is split off from the video and processed separately, then woven into the data stream in the final stage. The section “The DSP Audio Path” on page 11 describes the path for separately processed audio.

The purpose of this overview is

- ◆ To define the basic paths for video processing currently available in the 3DO development environment.
- ◆ To describe how to follow each of these paths from the initial capture through the post-processing and compression phases to the final export of the movie to 3DO file format.

Chapter Overview

Topics in this chapter include:

Topic	Page
Introduction	1
The Main Paths Available	2
The MPEG Path	5
The EZFlix Path	10
The DSP Audio Path	11

The Main Paths Available

What You Start With

Typically, you begin with one of the following:

- ◆ A video tape.

The material may have been originally recorded on video tape, or it may have been recorded on film and then transferred to video tape by means of the Telecine process.

The video tape may contain video only, or it may contain accompanying audio.

- ◆ One or more digital files.

Typically, you begin with digital files when your material consists of synthetically produced video sequences.

Most of the tools that produce this kind of video material allow you to save it as a QuickTime file.

If there is accompanying audio, the tools with which you created or edited it will have let you save it as an AIFF (Audio Interchange File Format) file.

The Overall Process

Your goal is to perform any required editing and enhancement of the material (e.g., adding special effects) and then convert it to a compressed data stream that can be played on a 3DO M2 machine.

The process consists of the following basic steps for both video and audio material:

1. Digitize the material if it is not already in digital form.
2. Edit the material.
Shorten it, resize it, add special effects, and so forth.
3. Compress the material to reduce the amount of storage space it occupies and, more importantly, the data rate required to play it.
4. Convert the material to a standard chunk format that can serve as input to the final datastreaming tool.
5. Weave the video and audio chunks together into a 3DO data stream that can be played on an M2 machine.

The Main Paths You can Take - MPEG or EZFlix (and DSP Audio)

The two main paths you can take are the MPEG encoding path and the EZFlix path. If your video is accompanied by audio, you will usually have to process your audio using a separate DSP audio path in parallel with the MPEG or EZFlix video path. The exception occurs when you use hardware MPEG encoding with minimal or no editing, in which case you can MPEG encode both the video and audio.

Factors in Choosing a Path

The first and most important step in video processing is to decide which of the video processing paths to pursue. There are three sets of issues you should consider in making this decision.

- ◆ M2 System Resources For Real-Time Playback
- ◆ Resulting Video Quality
- ◆ Availability of Encoding Tools

M2 System Resources For Real-Time Playback

One important set of issues for this decision is the comparative amount of M2 system resources consumed by the various real-time video and audio decoders from which you can choose. The key resources are

- ◆ CPU usage
- ◆ Memory usage
- ◆ DSP usage (in the case of audio)

For example, MPEG-1 Video generally requires greater memory usage than EZFlix, but it requires lower CPU usage because real-time MPEG-1 Video decoding hardware is built into M2.

Resulting Video Quality

The quality of the final video, as it appears in the M2 title, is a key issue. MPEG-1 Video encoded by a high-end MPEG encoder provides the best image in virtually all cases. However, if top-quality video is not a necessity, and access to high-end MPEG encoding tools is limited, EZFlix can provide a convenient solution

If you choose EZFlix, experiment with the codec to determine what it is best suited for. This will help you to optimize your shoot by avoiding material it does not handle well.

Note: *Unlike MPEG, EZFlix doesn't always produce "VHS quality" video.*

Availability of Encoding Tools

Not all MPEG-1 Video encoders are created equal. This is particularly true for MPEG-1 Video encoders. Generally speaking, the quality of any MPEG-1 Video sequence is primarily a function of the particular encoder that compressed it. Fortunately, a growing number of vendors and service bureaus now provide very high quality MPEG encoding tools and services.

If your source material is in a high-quality video format (such as a D1 or Betacam-SP tape), or in the form of an industry-standard-format computer file (such as a QuickTime movie), it is relatively easy to get your source converted to an MPEG-1 Video Elementary Stream. This is the necessary starting format to prepare MPEG video for M2 playback.

You can use the 3DO MPEGXpress Real-Time MPEG Encoding System product. You can use another vendor's MPEG encoding product. Or you can go to a video post-production house or other service bureau.

Note: *We strongly recommend that you choose or specify some specific parameters of MPEG-1 Video streams when you do your MPEG encoding. See the current Release Notes for details.*

If you do not have easy access to MPEG encoding hardware, another possible route to MPEG-1 Video encoding is to use Sparkle, the software-MPEG encoder included in the M2 Video tools folder. Sparkle is a public-domain MPEG encoding program made available to M2 developers with the permission of its author.

Bear in mind, however, that Sparkle is still in an advanced experimental stage. The first release does not include a technical feature called "rate control," a key feature of high-quality MPEG video encoders. This means a significant amount of experimentation and iteration is probably necessary to achieve high-quality MPEG video streams using Sparkle.

If memory is limited, and therefore the deciding factor, note that run-time decoding of EZFlx-compressed data uses far less memory than run-time decoding of MPEG-encoded data.

Diagram of Video Processing Paths

Figure 1-1 gives you an overview of the available methods for transforming your original material to a 3DO data stream. The steps and tools involved are discussed in more detail in the ensuing sections.

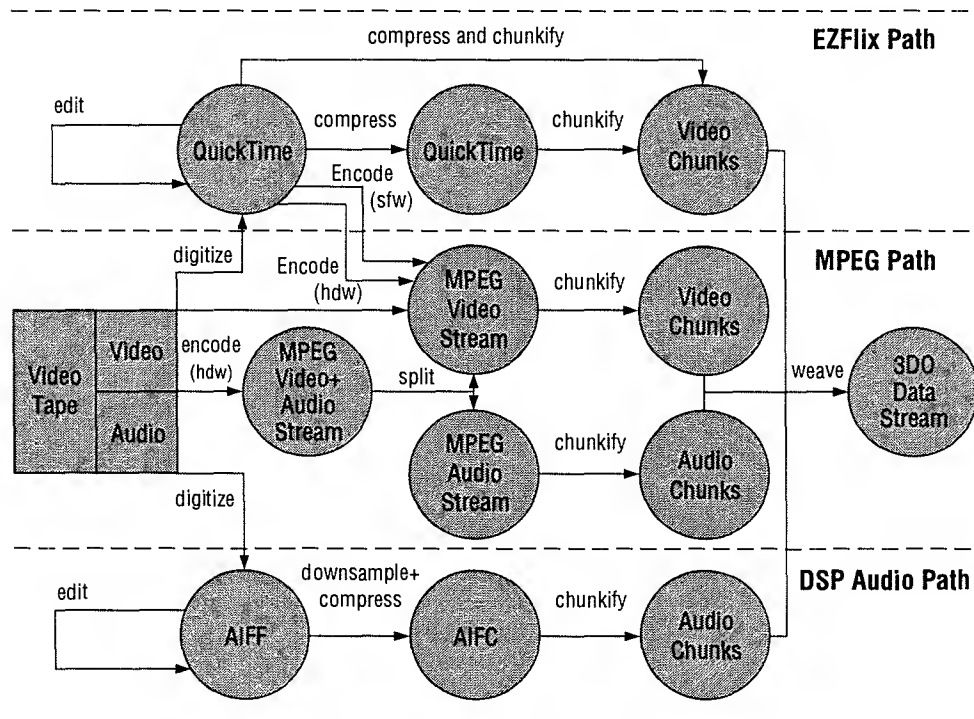


Figure 1-1 Overview of video processing paths

The MPEG Path

Overview

The exact procedure that you will use to MPEG encode your material will depend on the following factors:

- ◆ Whether you start with a video tape or a QuickTime file.
- ◆ Whether you have audio.

- ◆ Whether you need to edit the video or audio.
- ◆ Whether you have access to MPEG encoding hardware.

The sections below discuss the following typical scenarios:

- ◆ Hardware MPEG encoding with no editing.
- ◆ Hardware MPEG encoding with editing.
- ◆ Software MPEG encoding, using Sparkle.

Hardware MPEG Encoding With No Editing

If source material exists in final form on video tape and needs no post-processing other than the cropping and scaling available with most MPEG encoders, then video processing with MPEG hardware involves the steps shown in Figure 1-2 and described below.

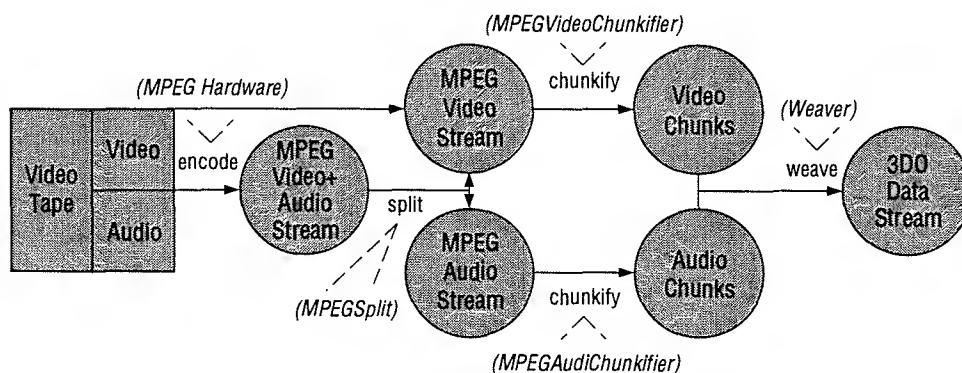


Figure 1-2 Hardware MPEG encoding with no editing

1. Encode/compress the video and audio data from the tape, using the MPEG hardware encoder.
2. Split the resulting stream into two separate streams, a video stream and an audio stream, using the MPEGSplit program.

If your data is video only, then you already have a video-only stream, and you can skip this step.

MPEGSplit is a public-domain Macintosh application provided to M2 developers with the Sparkle program.

3. Chunkify the video data, using the MPEGVideoChunkifier program, and the audio data, using the MPEGAudiChunkifier program.

MPEGVideoChunkifier and MPegaudioChunkifier are 3DO MPW applications provided to M2 developers.

4. Weave the data (video plus audio or video only) into a 3DO data stream, using the Weaver program.

Weaver is a 3DO MPW application provided to M2 developers.

Hardware MPEG Encoding With Editing

If you need to do more editing than the cropping and scaling available with most MPEG encoders, you will have to edit the video in a QuickTime file and the audio in an AIFF file as shown in Figure 1-3.

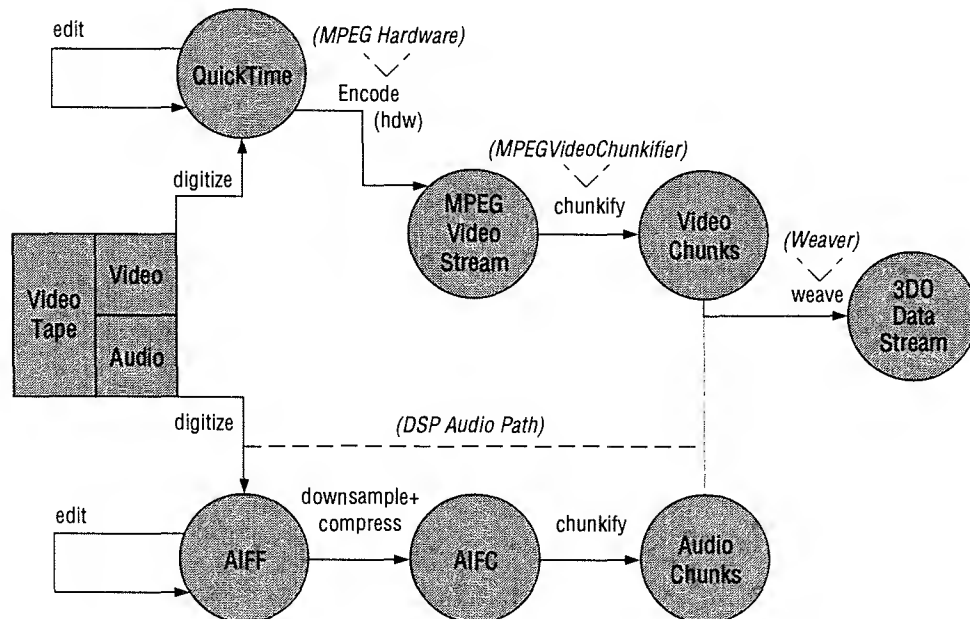


Figure 1-3 Hardware MPEG encoding with editing

1. Digitize the video data using a high-end digital capture system. This involves converting the analog data on the tape to digital data in a file.

The result for the video data will be will be a QuickTime file.

Material can be stored either in "raw" QuickTime uncompressed form or in compressed form using a lossless compression method.

If digitizing is done with the Animation QuickTime component, for example, capture should be made at Best Quality settings. This essentially creates an exact copy of the source material.

2. Edit/post-process the video data in the QuickTime file.

Postprocessing encompasses any modifications made to the digitized source movie before encoding and compression.

You would use such tools as 3DO MovieEdit, Adobe Premiere, Adobe AfterEffects, Equilibrium's DeBabelizer — in fact, any QuickTime manipulation program.

Postprocessing typically involves trimming out extra time, adding effects with filtering or, if the source was originally film, using 3DO 3-2 PullDown to reverse the telecine process (see "The 3-2 Pulldown Tool" on page 37).

3. Encode/compress the video data from the QuickTime file, using the MPEG hardware.

4. Chunkify the video data, using the MPEGVideoChunkifier program.

MPEGVideoChunkifier is a 3DO MPW application provided to M2 developers.

5. Digitize, edit, compress, and chunkify the audio data, using the process described in "The DSP Audio Path" on page 11.

Note that if you edit the video component via QuickTime, you must digitize the audio into an AIFF file and follow the DSP audio route -- even if you don't edit the audio. There is not currently a way to MPEG encode the audio component alone from a video tape and then recombine it with the video stream encoded from an edited QuickTime file.

6. Weave the chunkified video and audio data into a 3DO data stream, using the Weaver program.

Weaver is a 3DO MPW application provided to M2 developers.

Note that this is also essentially the same process you would use for MPEG encoding if your initial source material was already in the form of digital files (i.e., QuickTime and AIFF) rather than a video tape.

Software MPEG Encoding

Software MPEG encoding involves the same basic process as hardware encoding with editing except that the MPEG encoding itself is done by the Sparkle program rather than by MPEG hardware.

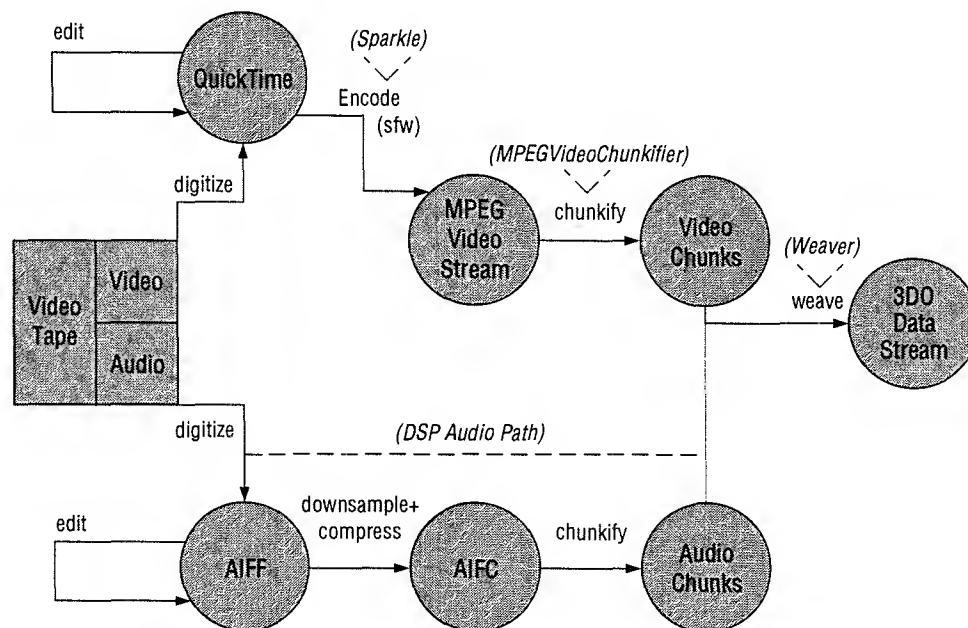


Figure 1-4 Software MPEG encoding using Sparkle

1. Digitize the video data into a QuickTime file.
2. Edit/post-process the video data in the QuickTime file.
3. Encode/compress the video data from the QuickTime file, using the Sparkle program.
Sparkle is a public-domain Macintosh application provided to M2 developers.
4. Chunkify the video data, using the MPEGVideoChunkifier program.
MPEGVideoChunkifier is a 3DO MPW application provided to M2 developers.
5. Digitize, edit, compress, and chunkify the audio data, using the process described in "The DSP Audio Path" on page 11.
6. Weave and export the chunkified video and audio data into a 3DO data stream, using the Weaver program.
Weaver is a 3DO MPW application provided to M2 developers.

The EZFlix Path

Video processing with EZFlix involves the procedures shown in Figure 1-5 and described below.

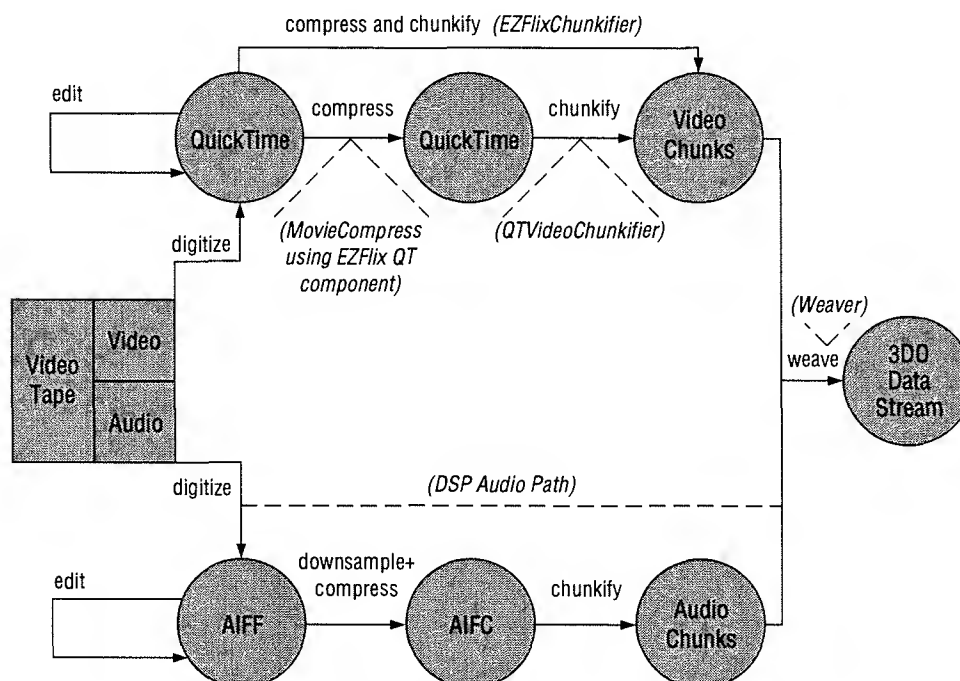


Figure 1-5 EZFlix processing

1. Digitize the video data into a QuickTime file.
2. Edit/post-process the video data in the QuickTime file.
3. Compress the video data from the QuickTime file, using the 3DO MovieCompress program, or any other QuickTime compression program, with the EZFlix QT component selected.
4. Chunkify the video data, using the QTVideoChunkifier program. QTVideoChunkifier is a 3DO MPW application provided to M2 developers.

An alternative is to use the EZFlixChunkifier program to compress and chunkify the video data in a single step.

5. Digitize, edit, compress, and chunkify the audio data, using the process described in "The DSP Audio Path" on page 11.

6. Weave the chunkified video and audio data into a 3DO data stream, using the Weaver program.

Weaver is a 3DO MPW application provided to M2 developers.

The DSP Audio Path

In most cases, you will process the audio separately from the video using the procedure shown in Figure 1-6 and described below.

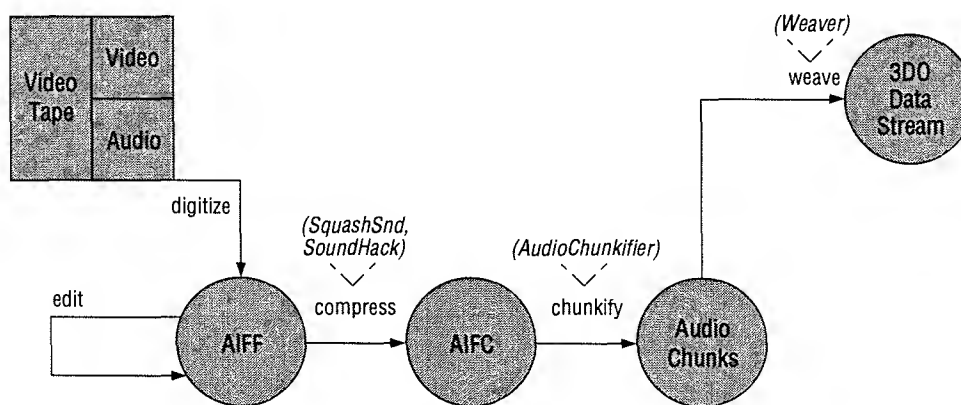


Figure 1-6 DSP audio processing

1. Digitize the audio with a high-end digital capture system.

The result of audio capture is an AIFF (Audio Interchange File Format) sound file.

2. Edit the audio data, using any AIFF-compatible sound editing tool.

Editing typically includes downsampling or sample rate conversion.

Downsampling is not a matter of dropping every other sample—the data must be refiltered. Tools that can do this correctly are:

- ◆ SoundHack 2.13
- ◆ SquashSnd 2.2a3

3. Compress the audio data into an AIFC sound file.

Use SquashSnd or SoundHack.

4. Chunkify the compressed audio data, using the AudioChunkifier program.

AudioChunkifier is a 3DO MPW application provided to M2 developers.

5. Weave and export the chunkified video and audio data into a 3DO data stream, using the Weaver program.

Weaver is a 3DO MPW application provided to M2 developers.

Digitizing

Introduction

This chapter provides information you need prior to starting the video digitizing process, then steps through the process itself.

Topics in this chapter include:

Topic	Page
Source Material	14
Digitizing Video	15
Processing Digitized Materials	17

Source Material

Video digitizing and compression are time-consuming, so prepare source material carefully.

Topics in this section include:

- ◆ Shooting Your Own Video
- ◆ Film-Based Source

Because each segment of video might be played many times during digitizing, the quality of the video can degrade significantly. Therefore, it is essential to make a copy of the source material for use as a “working master” and for dubbing working copies. Store the original in a safe place.

Typical source materials include:

- ◆ video tape
- ◆ film or video-based material from video tape or laser disc
- ◆ computer animations

See Chapter 3, “Compression,” to help you determine the best compression method for source material.

Shooting Your Own Video

Plan your shoot carefully. The following tips apply to shooting your own video:

- ◆ Determine the final frame rate before shooting so you can shoot at a lower frame rate, thus decreasing the disk space you’ll need.
- ◆ Reuse a single segment, if possible. For example, you might create a long sequence of running water by tiling together multiple copies of a single 10-second segment of running water.
- ◆ Use high-quality equipment, such as Betacam SP.
- ◆ Be prepared to deal with interlace effects in moving from analog (double-buffered) to digital video. You can deinterlace using linear interpolation with edge following, as discussed below.

Film-Based Source

Video derived from film requires special consideration for two reasons:

- ◆ The Telecine process inserts composite frames into the video.
- ◆ Film “noise” is enhanced by the compression process.

The Telecine Process

When producing video from film source material, studios use the Telecine process to transfer the source material from celluloid to magnetic tape, and from 24 fps to 30 fps. Telecine conversion is explained in detail in Chapter 5, "Video Tools," in the section, "The 3-2 Pulldown Tool" on page 37.

Eliminating Noise

When preparing film-based source material, keep in mind certain factors that can affect how well or how badly the material compresses. Most films have noticeable film grain noise and scratches. To the compression tool this noise is significant and is maintained and even enhanced during compression. Therefore, you should remove as much noise as possible before compression.

Some ways to reduce film grain noise include:

- ◆ Use a post-production studio to clean up analog film.
- ◆ If your source is film, reduce grain noise by using a median or recursive noise filter to pick out sharp anomalies, such as spots or pops.
- ◆ On D1, D2, and D3 video, use a digital filter right after the Telecine process.
- ◆ Because low-light shoots generate more film or video noise, use a three CCD camera (charge-coupled device) when shooting in low-light conditions.

Edge Crawl

Jagged edges cause edge crawl on TV monitors. Edge crawl can be caused by uneven resizing or aliasing, so always use anti-aliasing when resizing.

Other Considerations

Bear in mind that:

- ◆ A source with constant backgrounds and solid colors yields the best results.
- ◆ Slow pans/zooms are difficult to compress well (except with EZFlix).

Digitizing Video

Digitizing video is time consuming because almost all digitizing processes are multipass. Before digitizing, make sure you have a clean disk with enough space; digitizing to a fragmented disk causes problems.

You need the following three cards to digitize video:

- ◆ Video controller card
- ◆ Video acquisition card
- ◆ Audio acquisition card

The following sections recommend cards in these categories and briefly describes what they do.

Video controller card

A Diaquest card with Editmac and Quickpass is recommended.

The Diaquest card

- ◆ Controls two 422 devices
- ◆ Uses house sync for accuracy
- ◆ Uses VDIG or video card driver
- ◆ Tags QuickTime movies with time codes

With the Diaquest set up you

- ◆ Select the start frame
- ◆ Select the number of frames to capture
- ◆ Select size, compression, etc.

Video acquisition card

Choices include, among others, Video Explorer card, RasterOps card, Digital Film card, and NuVista card.

A video acquisition card lets you

- ◆ Digitize one frame of video in VRAM
- ◆ Use custom Plug-ins, VDIG
- ◆ Use selectable Gamma correction
- ◆ Choose among RGB, YPrPb, D1, and S-Video formats
- ◆ Take advantage of excellent quality RGB, analog-to-digital converter

Audio acquisition card

The following card lets you perform 16-bit audio acquisition:

- ◆ AudioMedia II: frame-accurate, uses MIDI.

Note: *To synchronize video and audio, you need to translate the video SMPTE time code to MIDI. You can use Video Time Piece hardware (made by Mark of the Unicorn) to do this.*

Digitizing Process

Generally you should get the highest-quality video tape possible and digitize at 640 x 480 in 24-bit color. It's best to start with the highest resolution because you lose detail later in compression. Furthermore, you will need 640 x 480 later to deinterlace properly. 640 x 480 requires multipass capture, which must be frame-accurate in order to synchronize audio.

Compression

When using a real-time capture board based on JPEG compression, set the JPEG compression level as “light” as possible to enable real-time capture. Otherwise JPEG can cause artifacts that will be compounded by EZFlix or Sparkle. Some early JPEG-based digitizing boards caused this problem no matter how light the compression setting.

Keep the following points in mind:

- ◆ Compressing while digitizing can degrade quality, but uncompressed digitizing uses considerable storage: $640 \times 480 \times 3$ (that is, R, G, B) == 1 megabyte per frame, about 1 frame per second.
- ◆ **Tape deck:** (Betacam, Hi 8) multipass capture is necessary but happens automatically; audio is added last.
- ◆ **LaserDisc:** Single-pass digitizing is possible.

Digitizing Audio

Digitizing audio is a separate process. You should digitize at 44.1 kHz or 22.05 kHz (not at the Apple standard— 22.25 kHz). If audio takes up more space than you can spare, subsampling from 44.1 kHz to 22.05 kHz is preferable to dropping from 16-bit audio to 8-bit audio.

Free-running audio digitizing results in loss of lip sync after 5 minutes. Frame-accurate capture (Audiomedia II card) requires SMPTE-to-MIDI time code converter but still results in loss of lip sync after 5 to 8 minutes. To achieve frame-accurate capture for longer periods of time requires an audio capture clock that is slaved to the video sync pulse.

Uncompressed Multipass Digitizing

When doing multipass digitizing with no compression, keep the following points in mind:

- ◆ Digitize as fast as you can write the output to the disk. Then rewind and get the next batch of input.
- ◆ 640×480 , no compression, 30 passes.
- ◆ 30 seconds of video takes about 30 minutes and uses about 1 gigabyte.

Processing Digitized Materials

After you’ve digitized your materials, further processing might be needed before compression, including:

- ◆ 3-2 Pulldown
- ◆ Deinterlacing
- ◆ Cropping and Resizing

3-2 Pulldown

Film-based material starts out at 24 fps. Insertion of composite frames during the Telecine process increases the rate to 30 fps. Because Telecine uses certain frames as the odd or even field of the video frame, removing these composite frames results in better-quality video.

3-2 Pulldown, for removing composite frames, is provided on the M2 Toolkit CDs. Additional documentation is covered in Chapter 5, "Video Tools." You might also use frame-by-frame comparison to remove composite frames in short video sequences.

Deinterlacing

True video is 60 fields sampled every second with two adjacent fields interlaced to construct a frame. When compressing 60-fps video use one field, either odd or even, and discard the other; that is, subsample each frame (captured at 640 x 480 at 30 fps) in the vertical direction. The resulting image is 640 x 240, with spatially aliased artifacts along diagonal lines in the vertical direction that appear as "edge crawl" on an NTSC screen. For some movies this is acceptable, but others will need edge reconstruction. Adobe After Effects now includes this function. Most digitizing routines will subsample if anything other than 640 x 480 is selected.

In summary, you should capture at 640 x 480 or 640 x 240, keep odd fields, and reconstruct even fields using Adobe After Effects or a comparable tool. Then use bicubic interpolation to reduce to 320 x 240.

Cropping and Resizing

If you want to change the frame size of your movie, remember that cropping is cheap and easy while resizing can be time consuming. And remember that you can create interesting effects by incorporating less than full-screen video into your title. For example, splitting the screen into different panels and showing video in only one of them can result in considerable space savings.

Cropping

Here are some values for cropping.

- ◆ Full frame
 - ◆ Frame buffer: 640 x 480
 - ◆ NTSC visible screen: 576 x 432
- ◆ Letter box
 - ◆ Frame buffer: 640 x 336
 - ◆ NTSC visible screen: 576 x 336

Resizing

Use Photoshop, DeBabelizer, or Adobe After Effects to resize. DeBabelizer and After Effects resize QuickTime movies best.

In general, resize by a factor of 50 percent. This blurs diagonal lines and reduces edge crawl. This means reducing 640 x 480 to 320 x 240, and 576 x 432 to 288 x 216.

Compression

Introduction

This chapter begins by comparing three video compression methods: EZFlix, Cinepak (called Compact Video in Macintosh documentation), and MPEG. Note that the discussion includes information about Cinepak for the sake of completeness although the 3DO M2 system does not currently support Cinepak. This chapter then looks at some quality issues.

Topics in this chapter include:

Topic	Page
Which Compression Algorithm	21
QuickTime and Cinepak Tools	24
Compression Options	25

For additional information about EZFlix, see Chapter 6, "Compressing with EZFlix." For details about software encoding with Sparkle, see Chapter 4, "MPEG Encoding with Sparkle."

Which Compression Algorithm

Video compression algorithms make choices that affect data rate, the quality of the compressed image, and the size of the resulting file.

This section compares the relative strengths and weaknesses of:

- ◆ EZFlix
- ◆ Cinepak (Not Currently Supported in 3DO M2)
- ◆ MPEG

EZFLix

EZFLix treats all frames as key frames and encodes pixels individually rather than together as rectangular blocks. This scheme works particularly well for some types of source material, while for others Cinepak might be a better choice.

This section discusses the advantages of compression with EZFLix versus Cinepak, and looks at some of the trade-offs involved.

All Frames are Key Frames

Treating all frames as key frames, that is, encoding every frame independently as if it were a still picture, minimizes the distortion of moving objects or backgrounds from one frame to the next that occurs in frame-differencing methods such as Cinepak. Any compression error or artifact in one frame does not propagate to future frames. Also, sudden changes (increases) in compressed data rate or image quality do not occur at scene-changes; and random access to any frame can be done quickly and easily.

Pixels are Encoded Individually

Encoding pixels individually rather than together as rectangular blocks produces random, distributed image quality artifacts rather than the annoying, block-based distortions that are the hallmark of Cinepak (*pixelization*—actually 2x2 or 4x4 blocks of pixels), and which occasionally appear even in good-quality MPEG (*mosquitoes*, which are caused when an object edge is not well-restored within the 8x8 pixel block.)

EZFLix artifacts resemble *snow* on a poorly received television channel. This snow is fairly unobtrusive when the source is real-world video (or film). The noise is more noticeable on flat, static, computer-generated backgrounds. Also, because every pixel is encoded separately, EZFLix can handle a full range of color and sharp detail—edges do not get blurred within blocks.

The relatively unobtrusive snowy artifacts of EZFLix, however, come at the cost of a fairly high data rate of around 1.7-2.2 bits/pixel. This translates to 190-240 KB/sec. for a 288x216 stream at 15 fps and a relatively high run-time CPU usage—probably 25-50% of 602 (estimated).

EZFLix generally looks very good on real-world video with lots of objects, background, and camera motion. If the camera is zooming, panning, or hand held, EZFLix is typically superior to Cinepak. If the sequence is composited, with frame after frame containing mostly the same static background, and only a small area of the screen in motion, Cinepak compresses better than EZFLix and can look as good or better if the objects and backgrounds are simple or flat.

Cinepak (Not Currently Supported in 3DO M2)

Cinepak encodes pixels in 2x2 or 4x4 blocks using a method known as Vector Quantization (VQ). VQ methods are generally very fast and simple to decode.

An advantage of Cinepak over EZFlix is that Cinepak is a frame-differencing method. However, Cinepak's frame differencing method is less sophisticated than MPEG's motion compensated method. Cinepak can take advantage of frame differencing only for pixel blocks that do not move from one frame to the next. This means that Cinepak must re-encode almost every pixel block in detailed backgrounds during camera panning.

Cinepak has difficulty with highly detailed images or images with lots of subtle color gradations. This is because Cinepak can hold only a limited number of block types in its codebook and must make frequent "best fit" approximations.

Cinepak performs best for simple, large-window video with primarily flat, static backgrounds and little detail or color range. On such streams Cinepak looks very good and requires a fairly low data rate (100-150 KB/s).

Cinepak can also compress complex video with lots of motion to low rates, but the quality may be unacceptable. Cinepak also performs poorly on small-window streams (which need all the resolution they can get) because Cinepak cannot use smaller than 2x2 blocks. EZFlix, by comparison, is a very good solution for small streams.

MPEG

Although MPEG is block-oriented, that is, it divides the entire image up into 8x8 tiles or "blocks," it uses the Discrete Cosine Transform (DCT) to encode each block. The DCT is an excellent transform because it can represent any amount of detail within a block as long as it uses enough bits in the encoding. In practice, even detailed pixel blocks compress very nicely and without distortion with the DCT. With a good MPEG Encoder, excellent quality can be achieved almost invariably at 150-200 KB/s for 320x240 streams at 30 fps. And "excellent quality" here means substantially better than EZFlix or Cinepak at their best. (Note that the comparable data-rate numbers given for Cinepak and EZFlix use smaller frames *and* half the frame rate.)

Like Cinepak (and unlike EZFlix), MPEG relies heavily on frame-differencing. A good MPEG encoder can do a very good job of efficiently encoding only the information that changes from one frame to the next. But unlike Cinepak, MPEG uses motion-compensation. This means that even when an 8x8 block moves from one frame to the next (due to camera panning, object motion, etc.), the block need not be re-encoded. Only its "motion vector," the amount by which it was displayed in X and Y, need be sent.

One disadvantage to decoding on M2 with MPEG versus EZFlix or Cinepak is memory usage. MPEG requires space for two reference-frame buffers in main memory. For 320x240 frames, this amounts to 225 kbytes of memory space. EZFlix and Cinepak require considerably smaller working spaces.

Another possible drawback to MPEG is the high complexity (CPU power) required to encode and decode MPEG streams in real time. But because MPEG decoding is built into M2, this is not a disadvantage in the M2 development environment. With a good-quality encoder, MPEG Video image quality and compression is superior to EZFlix and Cinepak in almost all cases.

Note: *EZFlix is provided on M2 primarily as a transitional codec on the way to an exclusively MPEG digital video encoding format for all M2 titles.*

QuickTime and Cinepak Tools

The Apple QuickTime tool, ConvertToMovie, is included on the QuickTime 2.0 CD-ROM. ConvertToMovie is available through AppleLink and Internet FTP access (the IP address is ftp.apple.com). The 3DO tool, MovieCompress, performs the same function as ConvertToMovie.

EZFlix and the QuickTime conversion tool, QTVideoChunkifier, are included in the *Streaming* folder in the Portfolio portion of the M2 CD.

This section discusses the following issues related to compression:

- ◆ Size Constraints
- ◆ EZFlix Advantages
- ◆ Setting Data Rate

Size Constraints

Bear in mind that storage becomes an issue as soon as you get into the digital world. Each second of video takes about 5 minutes to process and requires 30 MB when digitized. Don't expect to digitize first and edit later; instead, edit your beta tapes as far as possible before compressing them. Try to digitize only the material you need and use the highest quality possible.

EZFlix Advantages

Currently, you can achieve EZFlix compression by using MovieCompress, ConvertToMovie, Adobe Premiere, or DeBabelizer with the EZFlix QT component. While EZFlix compression can result in artifacts, it has the following advantages:

- ◆ It is very easy to use.
- ◆ You can use it along with any QuickTime editing tool.

- ◆ Relatively little memory is required at run time to decompress EZFlix-compressed data.
- ◆ Branching and looping are easy because each frame is self-contained.

Setting Data Rate

The Apple conversion tools provide the ability to specify an optimum data rate on which they base their compression. The playback target data rate is the *maximum* rate at which the compressed video data can be read from the CD-ROM drive. Although the CD-ROM drive in the 3DO Interactive Multiplayer has a theoretical maximum data rate of 300 kilobytes per second (for a double-speed drive), 200 - 250 kilobytes per second is the maximum range for video. Audio data requires between 40 and 100 kilobytes per second of the CD-ROM data stream.

Compression Options

To compress QuickTime source movies to QuickTime output, you can use almost any tool that does QuickTime compression, such as 3DO MovieCompress or Apple's ConvertToMovie. Use these two tools for compression only.

You can also use Adobe Premiere or Equilibrium's DeBabelizer, which are general-purpose QuickTime editing tools that also do compression.

Use 3-2 Pulldown to remove composite frames. Use AfterEffects or DeBabelizer to deinterlace to 20 or 30 fps, to edit (cut, copy, paste), and to resize.

For more information about when and how to use these tools, see Chapter 5, "Video Tools."

MPEG Encoding with Sparkle

Introduction

Sparkle is a freeware program that encodes QuickTime source into an MPEG elementary video stream. Encoding and preview is done completely in software on the Macintosh.

Overview

This chapter discusses MPEG encoding with Sparkle in the 3DO development environment. Although some discussion of data rate and other factors affecting quality is included, the chapter assumes a working knowledge of the MPEG standard and the data streaming process.

Topics include:

Topic	Page
Related Documentation	28
About Sparkle	28
Requirements	28
Preprocessing	28
Encoding with Sparkle	30
Putting MPEG into a Stream	35

Related Documentation

The *Sparkle Documents* folder in the *Video Tools* folder on the Toolkit CD contains all the Sparkle documentation written by the creator of the program. This chapter is essentially a distillation from those documents.

About Sparkle

Sparkle was written by Maynard Handley, who has given The 3DO Company permission to include it on the 3DO M2 Toolkit CDs.

Sparkle requires input in QuickTime form and produces an MPEG elementary video stream. To play the MPEG movie on the 3DO system, it must be exported to the 3DO format with the 3DO MPW shell DataStreamer tool, QTVideoChunkifier.

Audio must be processed separately and woven in at the end of the process with the Weaver, another 3DO MPW shell tool.

Requirements

Sparkle requires the following:

- ◆ A Power Macintosh with at least 12 MB of RAM.
- ◆ System 7.5.
- ◆ Plenty of hard disk space.

Note: *If your image comes from video tape or film (not a synthetic digital image), you need a video capture board or an AV Macintosh to get it into QuickTime format (see Chapter 2, "Digitizing").*

Preprocessing

For optimal quality, after deinterlacing composite frames in QuickTime movies digitized from film or video tape, break up the movie into segments of similar content. This enables you to configure MPEG encoding parameters separately for each type of sequence.

Use 3DO MovieEdit or any Adobe Premiere-like QuickTime editor to break up the video into sequences of several seconds each. Try to cut at clear transition points, such as fades between scenes.

If disk space is at a premium, you can use MovieCompress, Premiere, or any QuickTime compression tool to compress the video. We recommend using the Animation codec, which can be lossless at 100% but lossy at lower settings. The Video codec will almost always create some artifacts due to loss.

You are now ready to run Sparkle to create MPEG streams.

Some Considerations

Encoding with Sparkle is a five-step process in which some of the steps are repeated several times until you achieve acceptably high quality at an acceptably low data rate.

The five basic steps are as follows:

1. Open a QT movie segment
2. Save as MPEG
3. Fill out the MPEG encoding parameters and encode
4. Check the resulting data rate.

If the data rate is too high, go back to step 2, re-save the segment as MPEG, and then re-encode it with a higher compression (quantization) setting.

This will decrease data rate but also decrease quality.

5. Open the resulting movie and inspect its quality.

If the quality is too low, go back to step 2, re-save the segment as MPEG, and then re-encode it with a lower compression (quantization) setting.

This will increase quality but also increase data rate.

Within the framework of this basic procedure, a number of considerations come into play. Bear in mind, for example, that:

- ◆ Although full screen images at 15 fps or better is most common for the 3DO system, a smaller screen size or letterbox format can reduce the data rate and leave space for better audio. Encoding faster than 15 fps is problematic, however, because the data rate quickly exceeds the 300Kb/second CD limit. Note that the 3DO real-time MPEG encoder can do much better than this.
- ◆ Because MPEG is a flexible medium a wide range of MPEG stream quality is possible, with a constant trade-off between quality and data rate. For the 3DO Developer the maximum data rate is limited to 300KB/second, so some image quality must be sacrificed.

The encoding process involves a number of variables that can be altered to reduce data rate. Deciding which variables to adjust, and what adjustments to make, can be quite complicated because the encoding configuration depends on the content of the images being encoded and the specifics of the MPEG bitstream. For these reasons, MPEG encoding is an art form.

- ◆ Sparkle will not deliver the quality of a hardware encoding system. For the best possible MPEG quality for a given data rate, use a very good encoder (for example, the 3DO Real-Time MPEG Encoding System), because it can make some of the hard decisions for you.

- ◆ Because Sparkle currently does not support Rate Control, you cannot control the data rate of the resulting stream. Instead, you must manually lower the quality until you get an acceptable data rate for each movie clip. If you choose to use Sparkle, you should expect to do a lot of experimenting and iteration before you get acceptable results.
- ◆ You can preview the encoded MPEG using Sparkle, but ultimately the only way to verify the quality of clips, regardless of data rate settings, is to play them on a 3DO Player.

Encoding with Sparkle

This section describes how to set up Sparkle and encode a movie, and includes suggestions on balancing quality and data rate.

Topics include:

- ◆ Setting MPEG Playback Preferences
- ◆ Setting MPEG Encoding Parameters
- ◆ Fine Tuning

Setting MPEG Playback Preferences

Sparkle defaults are set for the fastest possible playback on the Macintosh. When encoding for the 3DO system, however, your priority is to obtain the best possible quality.

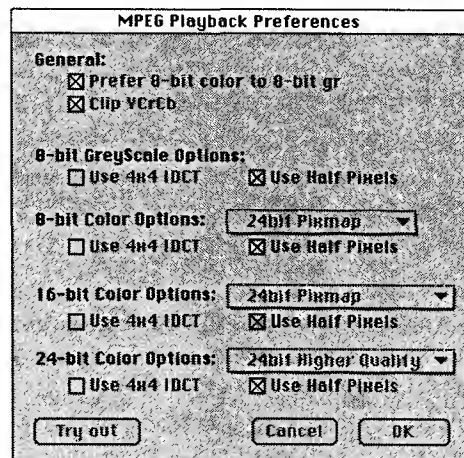


Figure 4-1 MPEG Playback Preferences.

When you launch Sparkle, immediately set the MPEG playback preferences for high quality decoding/viewing. Figure 4-1 shows the correct MPEG Playback Preferences settings for encoding for the 3DO system.

To set the Sparkle playback parameters, follow these steps:

1. Launch Sparkle.
2. From the Edit Menu, select "Preferences/MPEG Playback".

The MPEG Preferences dialog appears (see Figure 4-1).

3. Check all boxes *except* boxes labeled "Use 4x4 IDCT".

Leave these boxes unchecked because checking them results in better performance on the Macintosh but lower quality on the 3DO M2 machine. Your goal in this step is to get the best possible quality without worrying about performance on the Macintosh.

4. Set all pop-up menus to 24-bit Pixmap.
5. Click OK.
6. Select "Play Every Frame" from the Options menu.

This lets you play back the image and inspect each frame for quality.

Setting MPEG Encoding Parameters

Saving a movie as an MPEG stream is primarily a matter of filling out the MPEG Encoding dialog (Figure 4-2). Follow these steps to set MPEG encoding parameters:

Note: As noted in “Preprocessing” on page 28, on Quadras a bug in the “raw” codec causes some movies to play upside down. This is due to a specific QT call that Sparkle uses to render the image. If your movie displays upside down, you have a Raw QuickTime movie that cannot be used in Sparkle. To work around this problem, compress all your QuickTime movies—use MovieCompress and select the Animation codec at the highest quality settings for lossless compression.

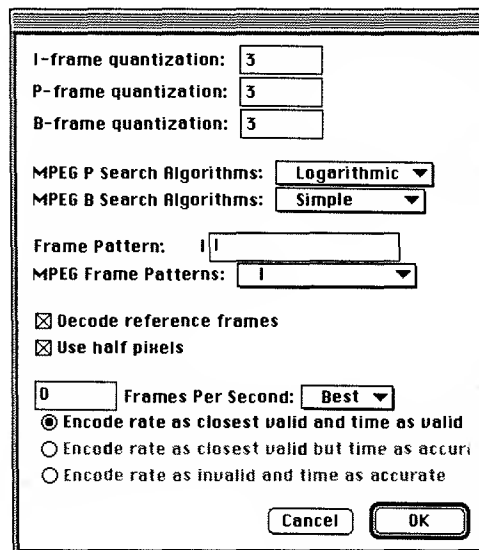


Figure 4-2 MPEG Encoding dialog.

1. Open a QT movie.
2. Select “Save As” from the File menu.

The Standard File dialog appears. At the bottom of this dialog box “File Type: MPEG Movie” is highlighted and “.MPEG” has been added to the end of the name of the QT movie.

3. Select a destination and click “Save”.

The MPEG Encoding dialog appears (Figure 4-2).

4. Fill out the MPEG Encoding dialog according to the settings in Figure 4-2, that is:
 - ◆ Quantization 1 to 4 for I, P, and B quantization settings
 - ◆ Default Search Algorithms settings
 - ◆ I, or PPPI Frame pattern
 - ◆ "Decode reference frames" and "Use half pixels" boxes checked
 - ◆ "Best" frame rate selected

Sparkle encodes at approximately the same speed as the Cinepak compressor. You can switch the encoder into the background, or pause it during encoding if you wish.

Note: See "ReadMe New Timing" in the Sparkle documentation folder for more information on frame rates. Unlike most QuickTime compression tools, which display the resulting compression image, Sparkle displays source images as it encodes them.

Fine Tuning

When the movie has been encoded, you can begin the process of

- ◆ Checking Data Rate
- ◆ Checking Quality
- ◆ Balancing Quality and Data Rate

Checking Data Rate

For playback on CD-ROM, the data rate of an encoded movie clip must be far enough below the 300K bytes/second limit (for double-speed CD-ROM players) to allow for all of the following:

- ◆ Audio playback at approximately 88 Kilobytes/second (for 16 bit 22K stereo).
- ◆ Plus any other data you might want to weave in.
- ◆ Plus padding bytes that are sometimes created by the Weaver to make data fit the available block size.

To calculate the data rate of an encoded clip, divide the size of the clip by the length (in seconds). To find out the size of a clip, from the Finder select Get Info, or select View-By-Size on your destination folder. You can also get the length of a clip with the MovieAnalyzer tool from the QuickTime CD.

This rate calculation gives just the *average* data rate for the clip, not the peak rate. It is quite possible that some frames are more complex than others and that the peak rate exceeds the CD bandwidth, which causes frames to be lost. Bigger stream buffers might help overcome this problem.

See the *3DO DataStreamer Programmer's Guide* for more information to help you get the data rate down to an acceptable level.

Checking Quality

The easiest way to check the quality of the MPEG movie is to open the movie and play it. By setting the Playback parameters to the highest settings, you can observe the kinds of artifacts introduced by MPEG encoding. These typically include a kind of blockiness and lack of detail in the resulting image.

When viewing an MPEG encoded movie on the Macintosh, bear in mind that:

- ◆ The image will be much bigger because most televisions have lower DPI than the Macintosh. Consider blowing up the image using the Grow Image menu item.
- ◆ Playback will be full speed
- ◆ NTSC hot colors can cause problems

The best way check the quality of an MPEG movie is to weave it into a stream and use the Video Player.

Balancing Quality and Data Rate

Getting the right trade-off between data rate and quality can be a problem. Obvious ways to lower the data rate include:

- ◆ Lower the Frame Rate
- ◆ Reduce the Frame Size
- ◆ Filter out noise or other non-essential details

The DeBabelizer application is helpful with all of these tasks, but the best approach is to reduce quality by adjusting the MPEG encoding parameters.

This can be done in two ways:

- ◆ Lower the Quantization levels (lower quality is achieved by using higher quantization values).
- ◆ Encode temporally

Lowering the Quantization levels (the straightforward approach) yields big savings in resulting disk size at the low end of the scale. For example, there is a big jump between Quant 1 and 2, and a fairly small jump between 5 and 6. Quant 4 works fairly well.

Encoding temporally yields good results on images that don't change a lot. Unlike Cinepak, MPEG searches for objects that have moved between one frame and the next. The pop-up menus of the Encoding dialog contain several algorithms, ordered fastest to slowest, for doing this search. The Sparkle technical notes explain how to do these searches.

Temporal encoding involves adding P and B frames. Adding P frames works quite well for most clips, but clips with lots of changes can grow larger using P frames. Adding B frames can be problematic and is not recommended because

some minor bugs in MPEG encoding using B frames can cause the encoding process to loop instead of terminate at the end of a clip (work around this by manually stopping the encoding process at the end). At lower quality, B-frame setting seems to introduce blurriness. This might occur only with the Half-Pixel option on, so experiment.

Putting MPEG into a Stream

To get an MPEG movie into a stream you must run MPEGVideoChunkifier and weave the result into a stream. However, you must first process your sound with an audio compressor such as SquashSnd or SoundHack. M2 supports several audio compression alternatives.

Given 44K-sampled 16-bit stereo as a starting point, you can get a factor of two compression by going to mono, down-sampling to 22K, or applying one of several 2:1 compression methods.

You can get 4:1 compression by using ADP4 compression or by combining two 2:1 methods; for example, down-sampling to 22K and using CBD2 compression.

You can get 8:1 compression by combining three 2:1 methods: for example, down-sampling to 22K, going to mono, and using SQS2 compression. This method is recommended because it delivers very good quality at an acceptable data rate, and the decoding is supported by hardware. This provides good sound for demo sequences and frees the processor for special effects.

Figure 4-3 is a summary of the alternatives that might be used to reduce the data rate for 44K-sampled 16-bit stereo sound:

Figure 4-3 *Compressing 44K-sampled 16-bit stereo sound*

Sound	Ratio Achieved	Data Rate
44K Stereo No compression	1:1	172K/sec
44K Mono No compression	2:1	88K/sec
22K Stereo No compression	2:1	88K/sec
44K Stereo CBD2 compression	2:1	88K/sec
44K Stereo ADP4 compression	4:1	44K/sec

Sound	Ratio Achieved	Data Rate
22K Stereo CBD2 compression	4:1	44K/sec
44K Mono SQS2 compression	4:1	44K/sec
22K Mono SQS2 compression	8:1	22K/sec (Recommended)

After running MPEGChunkifier on the MPEG stream, you must weave it with the audio. (See the *3DO DataStreamer Programmer's Guide* and the *3DO DataStreamer Programmer's Reference* in *3DO Tools for Programming I* for details.)

After weaving, you can play the resulting stream, using the Video Player, to see what the stream looks and sounds like on the 3DO system.

Video Tools

Introduction

This chapter describes the use of video processing tools available on the 3DO M2 Toolkit and, in addition, the use of a separately available Apple compression tool. The chapter covers the following topics:

Topic	Page
The 3-2 Pulldown Tool	37
MovieEdit	43
MovieCompress	45
EZFlixChunkifier	48

The 3-2 Pulldown Tool

The 3-2 Pulldown tool removes composite frames from a QuickTime movie. This increases the movie's perceived resolution while reducing the bandwidth required to display it on the 3DO Interactive Multiplayer.

Note: 3-2 Pulldown currently cannot handle unflattened QuickTime movie files. If you do multipass capture of source material, flatten the movie before processing with 3-2 Pulldown.

Only movies that start as film and are converted to analog video need 3-2 Pulldown processing. If the movie you are working with did not start as film, skip this section. If you're not sure whether your movie was originally captured from film, follow the steps in "Identifying Composite Frame Sequences in Edited Film" on page 41 to check for the composite frames present in movies converted from film to video.

Topics in this section include:

- ◆ Understanding Telecine
- ◆ Choosing a Movie
- ◆ Field Dominance
- ◆ Removing Composite Frames
- ◆ Saving a Processed Movie
- ◆ Troubleshooting

Understanding Telecine

Composite frames are intermediate frames inserted into a movie during Telecine conversion from film to video. This process converts film that displays at 24 frames per second into video tape that displays at 30 frames per second. Each set of four film frames is converted into interlaced video fields and then the video fields are reconstructed into five video frames as shown in the figure below.

The third and fourth video frames in this figure are composite frames that replace the second film frame. With these two composite frames included, the movie can be displayed at the 30 fps rate required by most analog video standards.

When analog video is converted into a QuickTime movie, however, these composite frames are included. Since the human eye often does not distinguish the difference between 24 fps and 30, you can remove the composite frames from your movie and display it at 24 fps in your 3DO application. This results in a clearer picture since the composite frames are not true digitized versions of corresponding film frames.

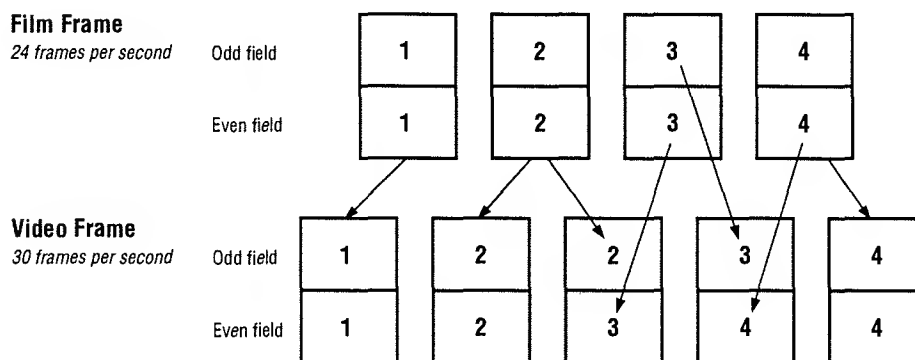


Figure 5-1 The Telecine process inserts frames.

Choosing a Movie

Currently, 3-2 Pulldown processes only QuickTime movies that meet the following criteria:

- ◆ 30 frames per second
- ◆ Raw format (no compression)

You can check the frame rate of a movie and find out whether it's been compressed with a QuickTime tool. Apples's MovieAnalyzer tool includes the frame rate and compression codec in the information window it displays for each video track in a movie.

Note: 3-2 Pulldown requires raw source data. Make sure your movie has not been compressed with JPEG before you process it with 3-2 Pulldown.

Field Dominance

To process a QuickTime movie correctly, you need to determine its field dominance. The field dominance setting determines whether a video capture system (video tape deck, laser disk recorder, etc.) selects an even or odd interlace field as the first field when it creates the first frame of a movie. Field dominance selections for two standard video capture systems appear below.

System	Field Dominance
Pioneer LaserRecorder	Even
Sony Betacam	Odd

To set the field dominance,

1. From the File menu, select Preferences.
The 3-2 Pulldown Preferences Dialog appears.
2. Select Odd or Even, depending on the field dominance of your movie. If you don't know what system was used for capture, choose Odd (the most common field dominance) and then check for a field dominance error when saving the processed movie. See "Incorrect Field Dominance" on page 42 to find out how to check for field dominance errors.

Removing Composite Frames

This section steps you through the process of removing composite frames from a movie. Topics include:

- ◆ Identifying Composite Frames
- ◆ Identifying Composite Frame Sequences in Edited Film
- ◆ Marking a Movie

When you open 3-2 Pulldown, the standard File dialog appears. Select the movie you want to process and click Open. The movie displays in a standard movie player window. You can step forward or backward by clicking on the frame-advance icons in the lower-right corner of the display or by using the left and right arrow keys.

Identifying Composite Frames

As you step through the movie, two out of every five frames should be composites. Because the composite frames are created from interlaced fields from two different film frames, you can identify them by the misalignment or “aliasing” of the interlaced fields. Below is an example of a sequence of frames.

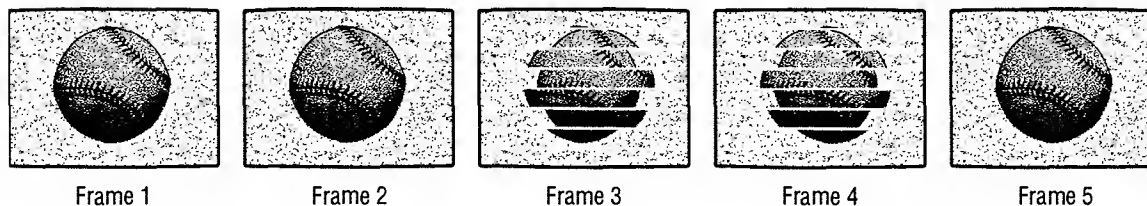


Figure 5-2 *Composite frames show misaligned fields.*

When you identify a pair of composite frames, mark the first one by choosing Mark from the Frame menu. Doing this identifies the composite frame as the first in a pair that will be converted into a reconstructed frame by 3-2 Pulldown.

A status bar under the movie window shows the time, frame, and phase numbers for the frame displayed. 3-2 Pulldown updates the status bar as you step through the movie. The green rectangle in the status bar represents a mark for that frame.

The phase counter represents the five-frame sequence of true and composited frames that 3-2 Pulldown expects. If the sequence of composite frames goes out of phase, return to the last correctly marked set of composite frames and review the movie again until you can identify and mark the next set of composite frames.

Identifying Composite Frame Sequences in Edited Film

With film that has been edited after Telecine processing, there can be three composite frames in sequence. This usually occurs when two sets of composite frames are combined during editing of a video tape. Of the three composite frames, the first two should be used to form one reconstructed frame, and the third frame should remain as an “orphan” frame.

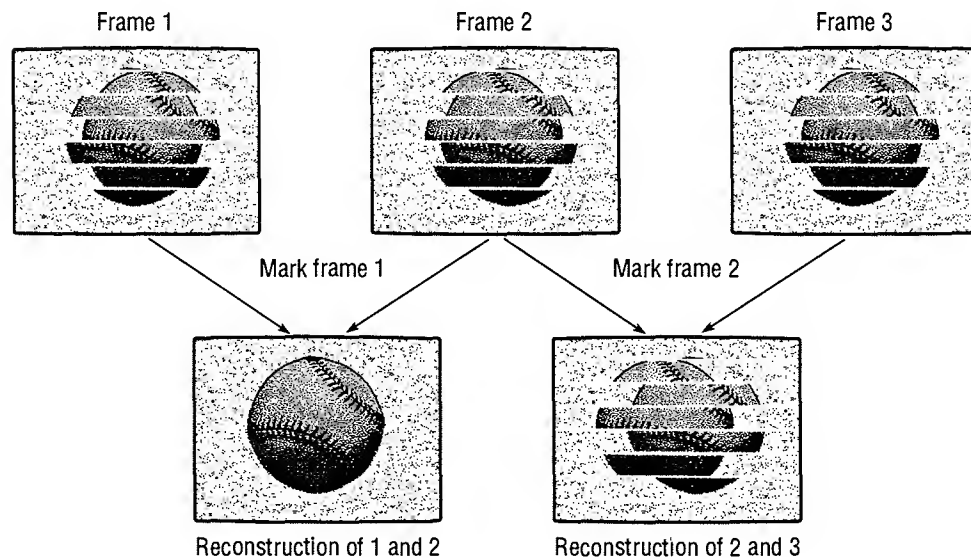


Figure 5-3 *Correctly and incorrectly marked composite frames.*

To accomplish this, be careful to mark the first frame, not the second one.

Figure 5-3 shows the result of marking the first frame as opposed to the result of marking the second frame. In this sequence,

- ◆ Marking frame 1 causes the correct set of composite frames (1 and 2) to be used, which results in a true reconstructed frame.
- ◆ Marking frame 2 causes the wrong set of composite frames (2 and 3) to be used, which results in an aliased reconstructed frame.

Marking a Movie

It may take several passes through a movie to mark all the composite frames correctly. Because edits may change the sequence of composite frames, you cannot assume that marking the first set of composite frames synchronizes the reconstruction process for the remainder of the movie. To speed up the review process, use the tab key to step through the marked frames.

Saving a Processed Movie

Choose Save as Pulldown from the File menu to save the movie. When 3-2 Pulldown saves, it displays the reconstructed frames as they are created. Check each reconstructed frame for the interlace field aliasing that indicates an incorrectly reconstructed frame. You might need to reset the frame markers and save the movie again.

Uncompressed QuickTime movies take up large amounts of disk space. Processing to remove composited frames requires enough disk space to store both the unprocessed and processed versions of the movie. Removing composited frames—that is, one out of every five—reduces the size of the unprocessed movie by approximately 20 percent and cuts the required disk space accordingly.

It is possible that a processed movie will contain fewer composited frames if it's edited after Telecine processing, resulting in a final file size greater than 80 percent of the number of pre-Telecine frames. The processed movie will never be larger than the unprocessed movie, however.

Troubleshooting

This section contains some troubleshooting tips on the following topics:

- ◆ Synchronizing Audio
- ◆ Incorrect Field Dominance

Synchronizing Audio

A processed movie should contain 80 percent of frames of the unprocessed movie. For example, a 900-frame movie should contain 720 frames after 3-2 Pulldown processing. If an unprocessed film contains more than the normal number of composite frames as a result of post-Telecine editing, the processed film may not contain the expected number of total frames. This can put the movie's audio track out of sync with the video track, and you may have to paste frames back into the video track to restore video and audio synchronization.

Incorrect Field Dominance

If you've chosen the wrong field dominance, the reconstructed frames created by 3-2 Pulldown will not be interlaced properly, that is, will be aliased to a greater degree than they were in the corresponding composite frames. To remedy this, change the field dominance in the Preferences dialog and save the movie again.

MovieEdit

The MovieEdit program lets you play and edit QuickTime movies. Topics in this section include:

- ◆ Saving Movies with MovieEdit
- ◆ MovieEdit User Interface
- ◆ Limitations
- ◆ Tips and Tricks

With MovieEdit you can cut, copy, and paste at all stages of processing. Dynamic information about time code, time units, and frame number allows more precise editing than some other tools.

You can also change the duration of any number of frames (from one to all) in a movie. To do this, choose *Change duration* from the *Edit* menu and in the dialog that appears, enter a new duration in QuickTime time units.

Note: Increasing a movie's time duration causes the movie play bar slider to extend beyond the boundary of the play bar by the amount of increased duration. When this occurs the movie does not play to the last frame, but begins to loop from the part that extends past the play bar boundary to the second to last frame of the entire clip.

Figure 5-4 shows an example of a MovieEdit window. Note the time code, QuickTime time units, and frame number information displayed in Figure 5-4.

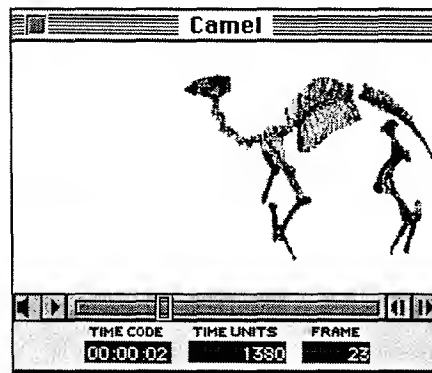


Figure 5-4 MovieEdit window.

Saving Movies with MovieEdit

MovieEdit lets you save movies in three different formats accessible through the Save As menu:

- ◆ **Dependent movie**—stores references to the movie resources. The original movie data files must be on the same Macintosh as the dependent movie file. This option saves smaller files than the original.
- ◆ **Self-contained movie**—stores the movie resources as well as the movie data. This movie does not need the original movie to be played. Saves larger files than dependent movie.
- ◆ **Cross-platform movie**—stores a self-contained, single-fork movie. This movie can be played on other OS platforms, for example SGI (UNIX) or PC (DOS) machines.

MovieEdit User Interface

MovieEdit provides menu-driven cut, copy, and paste functionality. Table 5-1 lists MovieEdit controls and keyboard accelerators.

Table 5-1 *MovieEdit functionality.*

Command	Keyboard Sequence
Start playing a movie	Double-click inside a movie window, or press Space bar or Return key.
Stop playing a movie	Click inside the movie window, or press Space bar, Return key, or Command-period.
Step frame forward	Right Arrow key.
Step frame reverse	Left Arrow key.
Increase/decrease volume	Click on speaker icon and an adjustable bar appears.
Turn sound on or off	Hold down Option key and click on speaker icon.
Jump to end	Option-Step frame forward.
Jump to beginning	Option-Step frame backward.
Play forward	Command-Step frame forward (Right Arrow key).
Play in reverse	Command-Step frame backward (Left Arrow key).
Manually control forward and reverse speed	Click and drag the slider control in either direction.
Loop	Hold down the Option key when you start playing.

Limitations

Note the following limitations in the current release of MovieEdit:

- ◆ No more than 20 movies can be open at one time.
- ◆ MovieEdit currently does not update time information dynamically.

Tips and Tricks

Cut or copy and paste segments of a movie only at frame boundaries. When using the slider to move to the end or beginning of a sequence you want to cut or copy, step one frame forward (or backward) to insure that you're at a frame boundary.

MovieCompress

The MovieCompress program lets you compress movies with EZFlx. With MovieCompress, you can specify

- ◆ frame rate
- ◆ key frame rate
- ◆ data rate
- ◆ depth
- ◆ codec

MovieCompress supports repeated compression of the same movie, and lets you run compression in the background so you can run another application while you wait for the movie to be compressed.

Topics in this section include:

- ◆ Movie Compression Settings Dialog Options
- ◆ Using the MovieCompress Tool

Movie Compression Settings Dialog Options

This section lists all Movie Compression Settings dialog options, with descriptions of each and recommended settings. These include:

- ◆ Compression Algorithm
- ◆ Color Quality Selection
- ◆ Compression Quality
- ◆ Frames Per Second

Compression Algorithm

The compression algorithm pop-up lets you choose how your source will be compressed.

EZFlix—The EZFlix compressor has the following characteristics:

Table 5-2 EZFlix compressor characteristics.

Characteristic	Discussion
Description	Fixed quality decompression scheme.
Recommended Sources	Anything with lots of motion or rapidly changing images.
Quality	Spatial compression only—ratios approximately 10:1.

Color Quality Selection

The Color Quality Selection pop-up differs depending on the compression algorithm you choose. Use it to choose between color and grayscale or to select the color depth to be used.

Compression Quality

The Quality slider lets you choose between high-quality/high data rate compression and low-quality/slower data rate compression.

Frames Per Second

The Frames per second pop-up lets you select a frame rate. You can also type in the desired frame rate. Whatever your source frame rate setting is, your destination rate should be the same or some fraction of that rate. That is, if your source is at 30 fps, your destination should be 30 fps, 15fps, or 10fps.

Note: For smooth displays, pick a frame rate that's half the original, because you must eliminate every other frame to end up with smooth transitions between frames. For example, if the original rate was 24, pick 12; if the original rate was 30, pick 15.

Using the MovieCompress Tool

This section illustrates the compression of a sample movie with EZFlix. The settings used in this example include:

- ◆ Compression: EZFlix, color
- ◆ Quality: variable
- ◆ Motion: frames per second at Best

To compress a movie, follow these steps:

1. Drag your source movie on top of the MovieCompress icon.

The Compression Settings dialog appears.

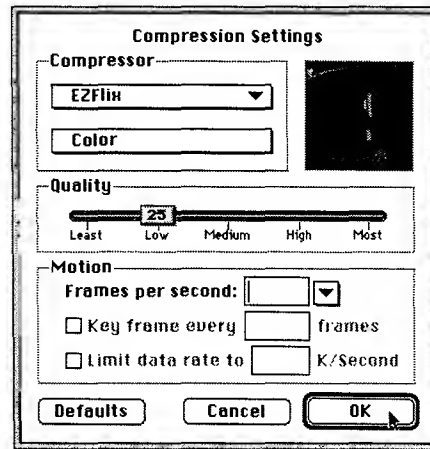


Figure 5-5 *Compression Settings dialog*

2. In the Compressor section, set the compression algorithm pop-up (Video by default) to EZFlx.
3. Select color.
4. You can move the Quality slider during compression to five different positions, which allows you to trade Image Quality against Data Rate. Table 5-3 shows the settings of the Quality slider and the expected data rate:

Table 5-3 *Quality slider settings.*

Quality Setting	Encoding Variables	Bits/Pixel Average	Minimum Bits/Pixel	Maximum Bits/Pixel
10%	3	1.8	1.125	2.25
25%	4	2.25	2.25	2.25
40%	4	2.0	1.125	3.375
60%	5	2.6	1.125	4.5
75%	6	2.9	2.25	4.5

Note: A Quality setting of 25% is the same compression you get using EZFlx in Opera.

5. In the Motion section, leave “Frames per second” at Best (the default setting). When you choose Best, MovieCompress preserves the movie’s number of frames per second.
6. Click OK.
A standard file dialog appears, prompting you for a destination file.
7. Specify the destination file.
A status dialog displays the progress of compression.

Note: *Using Adobe After Effects or Premiere is similar.*

EZFlixChunkifier

EZFlixChunkifier is an MPW Tool that does EZFlix compression (just as the EZFlix QuickTime component) does and chunkification (just as QTVideoChunkifier does) in a single step. EZFlixChunkifier is an Opera tool that has been updated to support M2 EZFlix compression as well.

Topics in this section include:

- ◆ EZFlixChunkifier Command-Line Arguments
- ◆ Weaving

EZFlixChunkifier Command-Line Arguments

The EZFlix compressor EZFlixChunkifier is an MPW tool with UNIX-style command-line arguments, listed in the following table.

Argument	Default Function
-c <number>	Specify the logical stream channel for the EZFlix chunks written to the output file. [Default = 0]
-i <pathname>	Specify the pathname of the input QuickTime movie file. [required]
-o <pathname>	Specify the pathname of the output EZFlix-chunk file. When no output file pathname is specified, the pathname of the input file concatenated with the “.EZFL” suffix is used.
-opera	Generate a file usable only for Opera EZFlix.

Argument	Default Function
-f <number>	Specify the frame rate of the output chunk file. Frames from the input QuickTime movie will be duplicated or dropped to meet this rate. [default = QT movie rate, or 15]
-h <number>	Specify the pixel height of each frame of the output EZFlix data. The frames of the input QuickTime movie will be cropped to achieve this result. [default = 192]
-q <number>	Specify the quality setting to be used for compression. The permissible values are 10, 25, 40, 60, and 75. The default setting is 25, which corresponds to Opera-compatible compression. Each higher quality setting yields a higher quality image, and a lower compression ratio that generates a higher data rate.
-v	Specify verbose mode, in which extra information is printed to the standard output. In particular, this mode will provide diagnostic messages if the encoder encounters any difficulties, such as extremely saturated pixels. Note: One message will be printed for each offending pixel encountered, so there is a possibility of swamping the standard output.
-w <number>	Specify the pixel width of each frame of the output EZFlix data. The frames of the input QuickTime movie will be cropped to achieve this result. [default = 256]

Table 5-4 Command-line arguments.

Weaving

The EZFlixChunkifier Tool does the EZFlix compression and generates a chunk file that is ready to weave. The 25% setting uses a fixed compression rate, so all frames will be the same size. The other settings are slightly variable, depending on the data being compressed. It is important to choose a Weaver BlockSize that allows several EZFlix frames to fit into one block without much filler leftover. Sound, and whatever else you put in the stream, should also be accounted for.

Compressing with EZFlix

Introduction

EZFlix is a 3DO video compression tool and playback engine for creating and playing back compressed digital video in software only. You can compress QuickTime movies, weave the compressed video with audio, and play the resulting stream with the EZFlixPlayer application.

EZFlix is targeted at near full-screen video.

About this Chapter

This chapter describes EZFlix functionality, and how to install the application and compress and play a movie.

Topics include:

Topic	Page
EZFlix Functionality	52
Installation	53
Compressing a Movie	53
Playing a Movie	54

Audience

EZFLix is for title developers familiar with QuickTime and the 3DO DataStreamer.

EZFLix Functionality

Some of the highlights of the EZFLix codec include the following characteristics and features:

- ◆ **Frame independence**—EZFLix compresses every frame individually. Every frame can therefore be decompressed without reference to any other frame.
- ◆ **Data rates**—Table 6-1 lists useful sizes and rates. These data rates are for the 25% quality setting, which has a fixed data rate. All other settings are somewhat variable and dependent on the content of the video.

The 40% setting normally provides a data rate about 10% lower than the 25% setting but with the same quality.

The 10% setting provides a data rate about 20% lower than the 25% setting but with a significant reduction in quality.

The 60% and 75% settings yield a data rate about 10% and 20% higher, respectively, than the 25% setting and provide somewhat higher quality video.

Table 6-1 *Frame sizes and rates.*

Aspect ratio	Frame size	Frame rate	Pixel rate	Data rate
~4:3	288x208	12fps	718,848	197 KBytes/s
4:3	256x192	15fps	737,280	203 KBytes/s
~5:3	288x176	15fps	760,320	209 KBytes/s

- ◆ **Image quality**—In most cases, EZFLix image quality ranges from very good to excellent. As with every video compression scheme, EZFLix adds its own characteristic visual “artifacts” to the original. While preserving edges and detail remarkably well, and rarely exhibiting any blockiness, EZFLix adds a mild form of random noise to the image.

While this effect often yields better image quality than even 16-bit (uncompressed) formats because it breaks up false contours, it also increases any noise already in the image. Therefore for best results, the source should be the “cleanest” 24-bit digitization possible.

If the source is synthetically generated, it should be rendered to 24-bit depth (“true color”). Also, because the added noise level is essentially scene-independent, best-looking results are obtained with good-contrast sources.

Note: *EZFlix does not use DSP.*

Caveats

When compressing QuickTime video with EZFlix, remember that a certain amount of color bleeding can occur across sharp, highly saturated color boundaries. This happens both with a highly saturated color against an unsaturated color (for example, screaming red against white), or with two highly saturated colors (screaming red against screaming yellow).

Also, EZFlix cannot represent extremely saturated colors, regardless of whether they occur on a sharp boundary. If the encoding tool encounters a pixel that is too saturated, it limits that pixel's saturation prior to compression.

Note, however, that image sources often contain saturated colors that are outside the composite video signal's palette. Off-the-shelf filtering tools (for example, the NTSC Video filter in Photoshop) can indicate whether a source contains colors that can't be displayed.

Installation

To install EZFlix, follow these steps:

1. In the *EZFlix* folder open the *Apps & Data* folder.
2. Copy the *EZFlixPlayer* application into your *remote* folder.
3. Copy the *EZFlix Component* to the *Extensions* folder and reboot.

Compressing a Movie

To compress a movie with EZFlix, follow this procedure:

1. Compress with *MovieCompress* and choose EZFlix compression.
2. Run the MPW tool, *QTVideoChunkifier* to create an EZFlix chunk file.
3. Create a weave script to weave the EZFlix chunk file and any audio file together into an EZFlix stream.

The Weave Script

The weave script can be specified in an MPW shell variable, and the weave step performed entirely from the MPW worksheet.

The following set of commands weaves an EZFlix stream from an EZFlix chunk file:

```
set WeaveScript 0
    'writestreamheader0n0
      streamblocksize 983040n0
      mediablocksize 20480n0
      streamstarttime 00n0
      streambuffers 40n0
      numsubsmessages 14000n0
      streamerdeltapri 60n0
      dataacqdelta pri 80n0
      subscriber EZFL 70n0
      subscriber CTRL 110n0
      file EZFlixExample.EZFL 1 00n'
echo "{WeaveScript}" | weaver -o "{3DORemote}EZFlixExample.stream"
```

To weave audio along with the EZFlix data, modify the script by specifying the audio chunk file and add directives to create the Audio subscriber and load the DSP instruments required to play the audio as follows:

```
set WeaveScript 0
    'writestreamheader0n0
      streamblocksize983040n0
      mediablocksize20480n0
      streamstarttime0 0n0
      streambuffers4 00
      numsubsmessages14000n0
      streamerdeltapri6 0n0
      dataacqdelta pri8 0n0
      subscriber EZFL 70n0
      subscriber CTRL 110n0
      subscriber SNDS 110n0
      preloadinstrumentSA_44K_16B_S_CBD20n0
      file EZFlixExample.EZFL1 00n0
      file Music.22K.Stereo.SAudio1 00n'
echo "{WeaveScript}" | weaver -o "{3DORemote}EZFlixExample.stream"
```

Note that the last three lines must be modified to suit your input files.

SA_44K_16B_S_CBD2

Playing a Movie

EZFlix lets you play back movies on a TV screen through the 3DO Development Station. The movie must be located in the *remote* folder.

To play a movie, follow this procedure:

1. Copy EZFlixPlayer and your movie stream (after the Weaver has run) to the *remote* folder
2. Launch the Debugger and in the Debugger Terminal window type:
`ezflixplayer <ezflixstreamfile>`
where `ezflixstreamfile` is the name of the movie.

Note: Check the 3DO Infoserver for upcoming postings of example streams.

Player Controls

The following controls let you manipulate movie play from the 3DO Joypad:

Table 6-2 *Player controls.*

Control	Action
P	Toggle between pause and play.
<- (left arrow)	Rewind the stream and play.
X	Stop the stream and exit.

Index

Numerics

- 3-2 PullDown VID-8
- 3-2 Pulldown VID-25
 - correcting incorrect field dominance VID-42
 - displaying reconstructed frames VID-42
 - identifying composite frames VID-40
 - overview VID-37
 - processing digitized materials VID-18
 - removing composite frames VID-40
 - saving processed movie VID-42
 - selecting field dominance VID-39
 - source requirements for using VID-39
 - synchronizing audio VID-42
- 3-2 Pulldown commands
 - Mark VID-40
 - Save as Pulldown VID-42
- 3-2 Pulldown tool VID-37
- 3DO MPEGXpress VID-4
- 3DO Real-Time MPEG Encoding System VID-29

A

- Adobe After Effects VID-18, VID-19
- Adobe AfterEffects VID-8
- Adobe Premier VID-8, VID-28
- Adobe Premiere VID-24
- AfterEffects VID-25
- AIFF VID-2, VID-7, VID-8, VID-11
- anti-aliasing resizing
 - edge crawl VID-15
- audio
 - capture VID-17
- audio acquisition card VID-16

- Audio Interchange File Format VID-2
- AudioChunkifier VID-11

B

- Best Quality settings VID-7
- Betacam SP VID-14

C

- CBD2 compression VID-35
- Cinepak VID-21, VID-22, VID-23
- Cinepak compression
 - characteristics VID-46
 - setting data rate VID-25
- Cinepak conversion tool VID-24
- composite frames
 - aliasing VID-40
 - created from interlace fields VID-40
 - in film edited after Telecine processing VID-41
 - marking VID-40
 - storage considerations VID-42
 - Telecine process VID-38
- compression
 - constant backgrounds VID-15
 - edge crawl VID-15
 - eliminating noise VID-15
 - MovieCompress Quality slider VID-46
 - pans and zooms VID-15
 - quality VID-46
 - solid colors VID-15
- compression options
 - MovieCompress VID-25

ConvertToMovie VID-24
 where to find VID-24
cropping VID-18

D

DataStreamer tool VID-28
DCT VID-23
DeBabelizer VID-19, VID-25
Debugger VID-55
deinterlace VID-25
deinterlacing VID-18
Diaquest card VID-16
Digital Film card VID-16
Discrete Cosine Transform VID-23
downsampling VID-11
DSP Audio VID-3
DSP Audio Path VID-11

E

edge crawl VID-15
 anti-aliasing resizing VID-15
EZFLix VID-3, VID-10, VID-21, VID-22, VID-24,
VID-51
EZFLixChunkifier VID-10, VID-48

F

field dominance VID-39
frame rate
 final VID-14
 selecting VID-46
frame size
 cropping VID-18
 resizing VID-19
frame-differencing VID-23

H

Handley, Maynard VID-28

M

master copy VID-14
MovieCompress VID-24, VID-28, VID-45, VID-53
 Color Quality Selection pop-up VID-46
 functionality VID-45

MovieCompress compression options
 Animation compressor VID-45
 Cinepak compression VID-46
MovieCompress options
 Frames per second pop-up VID-46
 Quality slider for choosing compression quality
 VID-46
MovieEdit VID-8, VID-28, VID-43
 dynamic information VID-43
 limitations VID-45
 saving cross-platform movie VID-44
 saving dependent movie VID-43
 saving self-contained movie VID-44
 window VID-43
MovieEdit functionality
 increase/decrease volume VID-44
 jump to beginning VID-44
 jump to end VID-44
 loop VID-44
 manual control speed VID-44
 play forward VID-44
 play in reverse VID-44
 start playing movie VID-44
 step frame forward VID-44
 step frame reverse VID-44
 stop playing movie VID-44
 turn sound on or off VID-44

MovieToStream
 where to find VID-24
MPEG VID-3, VID-5, VID-21, VID-23, VID-27
MPEG Encoding dialog VID-32
MPEG-1 Video Elementary Stream VID-4
MPEG-1 Video encoders VID-4
MPEGAudioChunkifier VID-6
MPEGChunkifier VID-35, VID-36
MPEGSplit VID-6
MPEGVideoChunkifier VID-6, VID-8, VID-9

N

noise
 compression considerations VID-15
NuVista card VID-16

P

Photoshop VID-19
preparing film source material VID-14

processing digitized materials
 3-2 Pulldown VID-18
 deinterlacing VID-18
processing paths
 diagram VID-5
 factors VID-3
 MPEG VID-5

Q

QTVideoChunkifier VID-10, VID-28, VID-53
QuickTime VID-2, VID-5, VID-7, VID-9, VID-10,
VID-16, VID-24, VID-27, VID-51
 Animation component VID-7
 animation component VID-7
QuickTime movie
 compression options VID-25
 edit with MovieEdit VID-43
 field dominance VID-39
 play with MovieEdit VID-43

R

RasterOps car VID-16
Rate Control VID-30
removing composite frames
 3-2 Pulldown VID-37
resizing
 QuickTime movie VID-19
 software for VID-19

S

sample rate conversio VID-11
setting data rate for Cinepak compression VID-25
shooting video VID-14
SoundHack VID-11
source material
 master copy VID-14
 preparing VID-14
 shooting video VID-14
 typical VID-14
Sparkle VID-4, VID-8, VID-21, VID-27, VID-28
SQS2 compression VID-35
SquashSound VID-11

T

Telecine VID-2, VID-14

Telecine process VID-15
 changing frame rate VID-38
 inserting composite frames VID-38
 understanding VID-15

V

VDIG VID-16
Vector Quantization VID-23
video acquisition card VID-16
video capture
 requirements VID-15
 video acquisition card VID-16
 video controller card VID-16
video capture process
 compression during capture VID-17
 requirements VID-16
video controller card VID-16
video digitizing VID-13
Video Explorer card VID-16
video formats
 Betacam VID-4
 D1 VID-4
video processing paths
 choosing VID-3
 factors VID-3
video processing tools VID-37
video tape VID-2
VQ VID-23

W

weave script VID-53
Weaver VID-7, VID-8, VID-11, VID-12, VID-28,
VID-49
weaving VID-36



3DO M2 FontBuilder User's Guide

Version 2.0 – May 1996

Copyright © 1996 The 3DO Company and its licensors.

All rights reserved. This material constitutes confidential and proprietary information of The 3DO Company. This documentation is subject to a license agreement with The 3DO Company and may be used only by parties to such agreement. Use by any other persons, and/or for any purpose not expressly authorized by the agreement, is strictly prohibited.

3DO's LICENSOR(S) MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. 3DO'S LICENSOR(S) DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME JURISDICTIONS. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

Other brand or product names are the trademarks or registered trademarks of their respective holders.

Contents

Preface

About This Document.....	FBR–vii
Audience.....	FBR–vii
How This Document is Organized.....	FBR–vii
Related Documentation	FBR–vii
Typographical Conventions	FBR–vii

1

Using M2 FontBuilder

Introduction.....	FBR–1
Chapter Overview	FBR–1
Creating an Anti-Aliased M2 FontBuilder Font	FBR–1
Saving Fonts.....	FBR–4
Using the “Other” Menu	FBR–5

List of Figures

Figure 1-1 M2 FontBuilder dialog.	FBR-2
Figure 1-2 Fontbuilder dialog showing minimum character range selection pop-up. .	FBR-3
Figure 1-3 FontBuilder Preview window showing completed font set.....	FBR-4
Figure 1-4 Get Font Info window.	FBR-5
Figure 1-5 Font Ascent/Descent dialog.	FBR-5
Figure 1-6 FontBuilder Character Spacing/Leading dialog.	FBR-6

Preface

About This Document

This document describes the M2 FontBuilder, a tool for creating anti-aliased fonts quickly and easily for the 3DO M2 system.

Audience

This document is for artists and font designers who prepare fonts for a 3DO title.

How This Document is Organized

This document consists of one chapter, "Using M2 FontBuilder" and an index.

Related Documentation

The 3DO FontBuilder allows you to create fonts, and display them on the Macintosh. Functions to display fonts are now part of the M2 Font Folio.

Typographical Conventions

The following typographical conventions are used in this book:

Item	Example
code example	<code>int32 OpenGraphicsFolio(void)</code>
procedure name	<code>CreateScreenGroup()</code>
new term or emphasis	A <i>ViewList</i> is a special case of a View.
file or folder name	The <i>remote</i> folder, the <i>demo.scr</i> file.

Using M2 FontBuilder

Introduction

The M2 FontBuilder converts Macintosh System folder fonts into anti-aliased fonts for use in the 3DO M2 development environment. In addition to standard Macintosh System fonts, you may also use outline fonts created with third-party applications, such as Fontographer, as long as you place them in the *Fonts* folder in your System folder.

You can preview the fonts you generate and save them in 3DO font file format. The M2 FontBuilder does not offer the ability to edit 3DO fonts. You can use a Macintosh font editor to modify the original outline fonts.

Chapter Overview

This chapter introduces the M2 FontBuilder. It covers:

Topic	Page
Creating an Anti-Aliased M2 FontBuilder Font	1
Saving Fonts	4
Using the "Other" Menu	5

Creating an Anti-Aliased M2 FontBuilder Font

To convert a font from your Macintosh System folder into an anti-aliased font with the M2 FontBuilder:

1. Double-click the M2 FontBuilder icon to open the application.

The FontBuilder dialog appears, as shown in Figure 1-1.

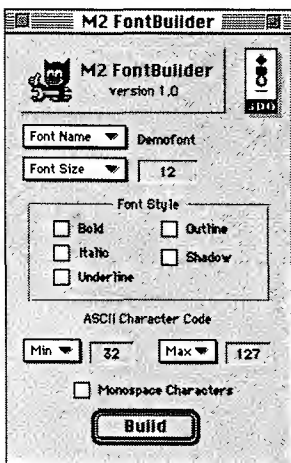


Figure 1-1 M2 FontBuilder dialog.

2. Select a font from the Font Name pop-up menu, which lists all fonts in the *Fonts* folder of your System folder.

Note: Many Macintosh fonts are copyrighted and may not be used in a 3DO title without proper licensing. A number of public domain fonts are provided on the 3DO Toolkit CD-ROM.

3. Use the Font Size pop-up to choose a point size or enter your own by selecting "Other..." from the Font Size popup menu.
4. Click on any Font Style checkbox (or combination of boxes) to modify the font's appearance.
5. Use the Min and Max pop-up menus in the ASCII character code section of the FontBuilder dialog (the Min pop-up is shown in Figure 1-2) to set the range of characters you want to include in your anti-aliased font set. While the Min or Max pop-up menu is selected, move the mouse to the alphanumeric or special character that will set the range of your font set and release the mouse button over the character.

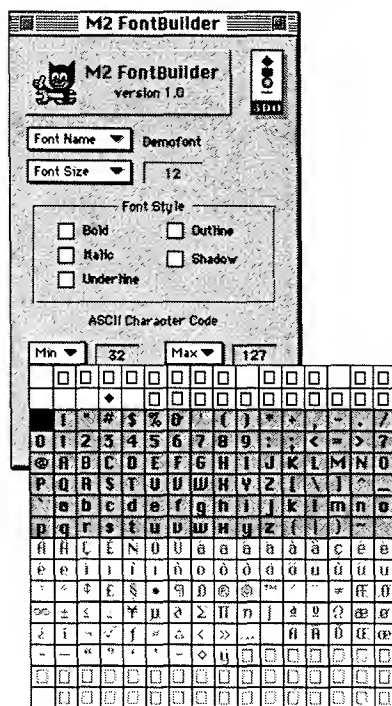


Figure 1-2 Fontbuilder dialog showing minimum character range selection pop-up.

6. Click the Monospace Characters checkbox if you want all characters in the font set centered within a space based on the width of the widest character.
7. Click the "Build" button at the bottom of the M2 FontBuilder dialog to generate a 3DO font. This font may be previewed on the Macintosh by clicking the zoom box in the upper right hand corner. The zoom box toggles preview mode on and off. Preview mode is on by default.

Note: You can watch the process of 3DO font conversion while it is taking place by leaving the preview window open.

A progress dialog appears, and characters begin to appear in the preview window. The preview area shows the font in different build stages during this process. Figure 1-3 shows the window after the font has been built.

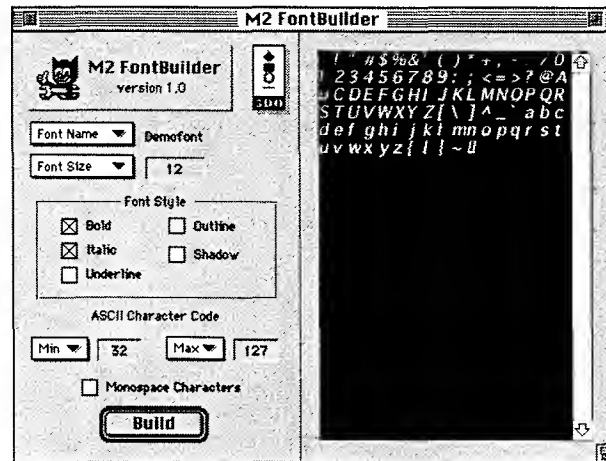


Figure 1-3 *FontBuilder Preview window showing completed font set.*

Saving Fonts

Follow the steps below to save an anti-aliased FontBuilder font that can be displayed on the 3DO system:

1. From the File menu, choose either Save or Save As.
2. In the file selection dialog that appears, provide a name for the saved font file and select Save.

Eventually you'll need to place this file in the *3do/remote* folder in order to use it with the 3DO System.

Using the “Other” Menu

After building a font, you can select Get Font Info (Command-I) in the Other menu for a summary of information about the font, as shown in Figure 1-4.

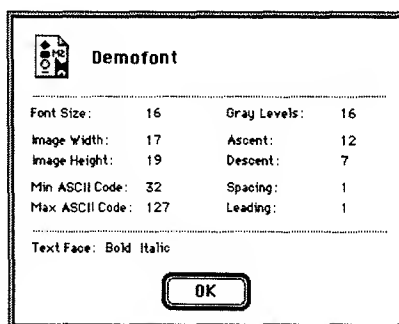


Figure 1-4 Get Font Info window.

There are two further options available for modifying the font at this point:

- ◆ To change the ascent or descent settings that are stored in the header of the font file, choose Select Ascent and Descent from the Other menu.
- ◆ In the dialog that appears, type in the settings you wish.

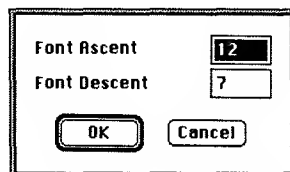


Figure 1-5 Font Ascent/Descent dialog.

3. To change the interspacing and leading values that are stored in the header of the font file, choose Set Spacing and Leading from the Other menu.
4. In the dialog that appears, enter the setting you prefer.

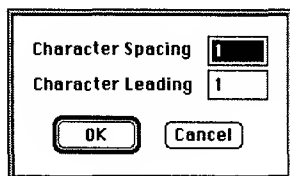


Figure 1-6 *FontBuilder Character Spacing/Leading dialog.*

5. Use the Save Font or Save Font As command to save the modified font information.
6. Clicking on Get Font Info will also show any changes to the font information.

Note: *Changes to font Ascent, Descent, Spacing, and Leading only affect the font descriptor information saved in a 3DO font file header. These values may change how the Font Folio displays a font by default in a 3DO title, but do not change the view in the M2 FontBuilder preview window.*

Index

A

anti-aliased font
 creating FBR-1
Ascent/Descent dialog FBR-5

F

Font Size pop-up menu FBR-2
Font Style checkbox FBR-2

G

Get Info command FBR-5

M

M2 FontBuilder FBR-4
M2 FontBuilder dialog FBR-1
M2 FontBuilder Preview window FBR-4
Min/Max pop-up menu FBR-2
Monospace Characters checkbox FBR-3

O

Other menu FBR-5

S

Save anti-aliased font FBR-4
saving fonts FBR-4
Set Leading and Spacing dialog FBR-5



3DO M2 CD-ROM Mastering Guide

Version 2.0 – May 1996

Copyright © 1996 The 3DO Company and its licensors.

3DO and the 3DO logos are trademarks or registered trademarks of The 3DO Company. All rights reserved. This material constitutes confidential and proprietary information of The 3DO Company. This documentation is subject to a license agreement with The 3DO Company and may be used only by parties to such agreement. Use by any other persons, and/or for any purpose not expressly authorized by the agreement, is strictly prohibited.

3DO's LICENSOR(S) MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. 3DO'S LICENSOR(S) DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME JURISDICTIONS. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

Other brand or product names are the trademarks or registered trademarks of their respective holders.

Contents

Preface

Audience.....	CDM-xi
How This Document is Organized.....	CDM-xi
How to Work With This Document.....	CDM-xii
Typographical Conventions.....	CDM-xii

1

Hardware and Software Requirements

Requirements for a 3DO M2 Development System	CDM-2
3DO Development System Hardware Requirements	CDM-2
3DO Development System Software Requirements	CDM-3
Requirements for a CD-ROM Recording Station	CDM-3
Recording Station Layout	CDM-3
Hardware Requirements	CDM-4
Software and Media	CDM-4

2

Preparing for the Mastering Process

Application Size Limitations	CDM-6
Factors Influencing Available Space	CDM-6
Determining Available Space for Your Application	CDM-6
Programming Notes.....	CDM-7
Using Relative Pathnames	CDM-7
Hints on Error Checking and Other Advice	CDM-7

3

Creating and Testing the Image File

Background and Overview	CDM-10
Layout Tool Background Information	CDM-10
Optimized Layout Process Overview	CDM-12
Required Files and Folders	CDM-12
Preparing Your Title's Files for the Layout Process	CDM-12
Modifying the BannerScreen File	CDM-13
Preparing Simple cdrom.image Files	CDM-13
Editing the cdrom.tcl File for the Simple Image	CDM-14
Creating the Simple Image	CDM-15
Testing the Simple Image	CDM-15
Preparing an Optimized CD-ROM Image.....	CDM-16
Creating the Base Image for Information Collection	CDM-16
Preparing the Debugger for Information Collection	CDM-16
Running Your Title to Collect Access Information	CDM-17
Running the Layout Tool to Create an Optimized Image	CDM-17
Testing the CD-ROM Image File on a Macintosh.....	CDM-20
Layout Tool Troubleshooting	CDM-20

4

Mastering and Testing the CD-ROM

Preparation and Setup for the Mastering Process	CDM-24
Maximizing Recording Speed	CDM-24
Setting Up the QuickTOPIX Software	CDM-24
Creating a CD-ROM Disc for Testing	CDM-27
Testing the CD-ROM Disc Using the Debugger.....	CDM-28
Debugging Hints	CDM-28
Creating a Master CD-ROM Disc.....	CDM-29

5

Delivery for Authorization

About the Authorization Requirements	CDM-32
--	--------

A

CD-ROM Mastering Checklist

Reading List.....	CDM-33
Preparing for the Mastering Process	CDM-33
Creating and Testing a Simple CD-ROM Image File	CDM-34

Creating an Optimized Image.....	CDM-34
Testing the Optimized CD-ROM Image on the Macintosh	CDM-36
Mastering the CD-ROM Disc.....	CDM-36
Testing the CD-ROM.....	CDM-38
Preparing Materials for Authorization	CDM-38

B

CD-ROM Mastering Etiquette

CD-ROM Basics	CDM-40
Constant Linear Velocity	CDM-40
3DO Drive Specifications	CDM-40
Data Rate Issues	CDM-41
Error Detection and Correction	CDM-41
Programming Do's and Don'ts	CDM-42
Making Assumptions about Data Delivery Rates	CDM-42
Handling Data Delays	CDM-42
Alternating File Reads	CDM-42
Testing Performance	CDM-43
Other Things You Should Do	CDM-43
Head Seeks and CD-ROM Mechanism Lifetime	CDM-44
Optimization Issues	CDM-44

List of Figures

Figure 1-1 3D0 M2 development system.	CDM-2
Figure 1-2 CD-ROM recording station.	CDM-3
Figure 4-1 QuickTOPIX setup screen for block image format.	CDM-25
Figure 4-2 QuickTOPIX setup screen for Apple HFS format images.....	CDM-26
Figure 4-3 QuickTOPIX dialog for creating a block image creation.	CDM-27
Figure 4-4 QuickTOPIX dialog for creating a CD-ROM for delivery.	CDM-29

List of Tables

Table 3-1 Variable settings for simple image creation.	CDM-14
Table 3-2 Variable settings for optimized image creation.	CDM-18
Table 5-1 Requirements for Authorization.	CDM-32
Table A-1 3DO drive specifications.	CDM-40

Preface

This document tells you how to move a working title to a CD-ROM disc which can be played in stand-alone mode on a 3DO™ M2 system. Using this information, you can create and test a CD-ROM image file and master an optimized CD-ROM for delivery to Matsushita Electronics Corporation (MEI) for approval.

Audience

This document should be read by the programmer developing the title *as well as* the person mastering the CD-ROM.

- ◆ Programmers must consider the effects of the mastering process while developing the title.
- ◆ The person who masters the CD-ROM can use the information in this document and work with the enclosed checklist.

How This Document is Organized

- ◆ Chapter 1, "Hardware and Software Requirements," discusses hardware and software requirements, including recommended CD pressing equipment and media.
- ◆ Chapter 2, "Preparing for the Mastering Process," explains preparing your application to run in stand-alone mode, testing your application in the minimum memory configuration, and testing for low memory pointers.
- ◆ Chapter 3, "Creating and Testing the Image File," steps you through the process of creating an optimized *cdrom.image* file and testing it on your Macintosh.
- ◆ Chapter 4, "Mastering and Testing the CD-ROM," describes how to master a CD-ROM with the QuickTOPIX software, and how to test your master using the 3DO Debugger and a 3DO M2 Development Station.

- ◆ Chapter 5, “Delivery for Authorization,” briefly discusses delivery to MEI for authorization.
- ◆ Appendix A is a CD-ROM mastering checklist.
- ◆ Appendix B, “CD-ROM Mastering Etiquette,” provides useful information about issues that you should pay attention to as you prepare for mastering your CD-ROM.

How to Work With This Document

The following approach is particularly effective when working with this document:

1. Make a copy of Appendix A, “CD-ROM Mastering Checklist” and check off each step as you proceed. If you have questions about any of the steps, refer to the more detailed instructions in the earlier chapters.
2. Make additional copies of the checklists as you prepare additional CD-ROMs.

Typographical Conventions

The following typographical conventions are used in this book:

Item	Example
code example	<code>Item CreateRenderBuffer(int rb_Size)</code>
procedure name	<code>ModifyGraphicsItem()</code>
new term or emphasis	In M2, <i>characters</i> are objects that can be displayed on the screen.
file or folder name	The <i>remote</i> folder, the <i>demo.scr</i> file.

Hardware and Software Requirements

Before you start the CD-ROM mastering process, make sure you have all required hardware and software. This chapter lists all requirements, providing options where available.

This chapter covers the following topics:

Topic	Page
Requirements for a 3DO M2 Development System	2
Requirements for a CD-ROM Recording Station	3

Requirements for a 3DO M2 Development System

To create a CD-ROM image file, you need a 3DO M2 development system. This section describes the hardware and software requirements for that system.

3DO Development System Hardware Requirements

Figure 1-1 shows the 3DO M2 Development System setup for creating the CD-ROM image file.

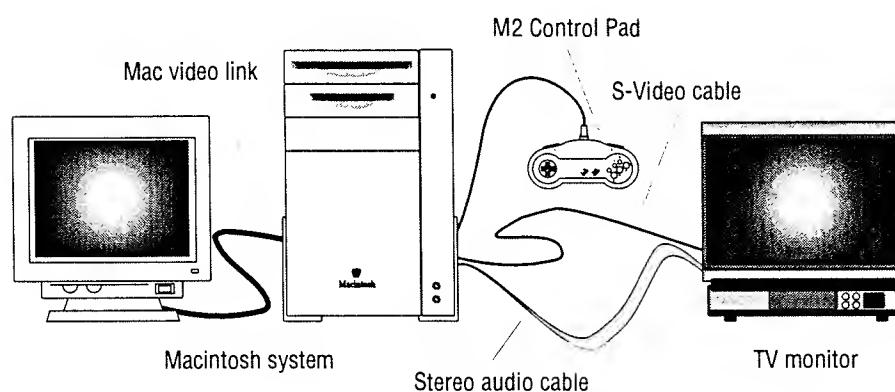


Figure 1-1 3DO M2 development system.

Here's a list of the required and recommended hardware:

- ◆ **Macintosh:** 68040-based or Macintosh based 6XX PowerPC CPU. Choose the 7100 or 8100 because they can fit a full sized (or long) NUBUS card.
- ◆ External hard drive with double-shielded SCSI-to-Macintosh cable (for moving CD-ROM image file—recommended but not required)
- ◆ **3DO M2 Development System**
 - ◆ 3DO M2 Development Card installed in a 68K or PPC Macintosh, and 3DO External CD-ROM drive (for testing only)
- ◆ TV monitor connected to the 3DO M2 development card inside the Macintosh

Note: See the manual, "Getting Started With 3DO M2 Release 2.0" for more information about the required hardware installation.

3DO Development System Software Requirements

You need the following software to create the *cdrom.image* file:

- ◆ Most recent 3DO M2 software
- ◆ MPW (Macintosh Programmer's Workshop)

Requirements for a CD-ROM Recording Station

You need a CD-ROM recording station to master the CD-ROM disc. This section lists hardware requirements and provides information about recommended recorders and media.

Recording Station Layout

Figure 1-2 shows one possible setup for the system on which you can master the CD-ROM disc.

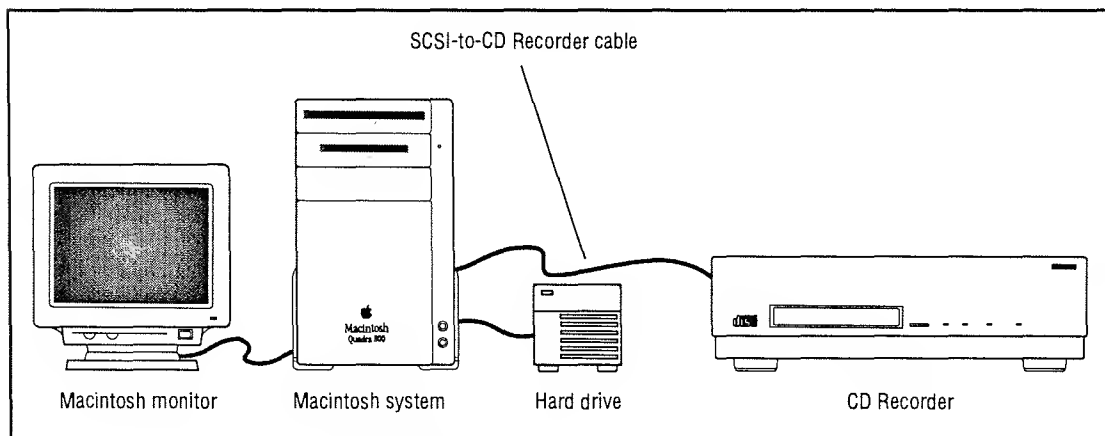


Figure 1-2 CD-ROM recording station.

Hardware Requirements

Here's a detailed list of all required and recommended hardware and other parts:

- ◆ 68040 or PPC-based Macintosh with monitor, keyboard, and mouse
- ◆ External hard drive (recommended for moving CD-ROM image file)
- ◆ Compact disc recorder
- ◆ SCSI-to-Mac cable
- ◆ SCSI-to-CD recorder cable
- ◆ Canister of compressed gas (recommended)

Note: *Use double-shielded SCSI cables.*

Recommended Recorders

3DO Developer Technical Support has had good results with Philips, Sony, Yamaha, and Kodak CD recorders.

There have been problems noted in the past when recording CD-ROMs with JVC and Pinnacle Micro recorders.

Software and Media

This section lists recommended software and media.

Software

There are a number of software packages available for the Macintosh that are capable of recording raw images, including QuickTOPIx2 (or later) and Astarte's Toast-CD. You can use other software packages as long as it is capable of recording raw image files.

Media

3DO Developer Technical Services has tested the following 74-minute media, although others will also produce satisfactory results:

- ◆ Taiyo Yuden: That's CD-R 74Q
- ◆ 3M: Type 74 media
- ◆ Ricoh: Type 74 media
- ◆ Verbatim: CD-R 74/63 4X compatible media

Preparing for the Mastering Process

This chapter helps you prepare your title for the CD-ROM mastering process.

Note: *Programmers are urged to read this chapter early in the title development process to prevent problems when the time for mastering arrives.*

This chapter covers the following topics:

Topic	Page
Application Size Limitations	6
Programming Notes	7

Application Size Limitations

This section helps you work with application size limitations and discusses the following topics:

- ◆ Factors Influencing Available Space
- ◆ Determining Available Space for Your Application

Factors Influencing Available Space

The amount of information that a CD-ROM discs can contain varies, but 650.4 MB is typical. This must include:

- ◆ The title's program and art files
- ◆ System files
- ◆ CD-ROM access information generated during the CD layout process

In most cases, your title can use approximately 600 MB on the disc. The exact portion of the 640 MB for your application data is variable and depends in part on these factors:

- ◆ How much space on the disc does the system require?
- ◆ How large are the individual developer files?
- ◆ How many avatars (instances) of the root, label, and directory files does the layout tool place on the disc?

Default numbers are: 3 directory avatars, 7 root avatars, and 2 label avatars. The *cdrom.tcl* file contains information on how to safely change the defaults.

Determining Available Space for Your Application

The only reliable way to determine exactly how large an application can fit on the disc is to run through the normal layout procedure described briefly below and in detail in the section "Preparing Simple cdrom.image Files" on page 13:

- ◆ Place the executable and data files in the */cdrommaster/takeme* folder
- ◆ Set the *cdrom.image* file size in the *cdrom.tcl* file to 640
- ◆ Run the layout tool

You can include the maximum amount of data when, in the *cdrom.tcl* file, image file size is set to 640 and the number of directory, root, and label avatars are all set to two. If you do this and the layout tool produces the error "Could not assign avatars for ...," you need to reduce your title's content.

Programming Notes

This section discusses some frequently encountered problems. It covers:

- ◆ Using Relative Pathnames
- ◆ Hints on Error Checking and Other Advice

Using Relative Pathnames

Applications should always use pathnames relative to the location from which the application boots.

For example, during development an application boots from */remote*. If artwork is located in */remote/art....*, the program should refer to those files as *art/....* During development, the pathname is then resolved to */remote/art* since the program booted from */remote*.

Warning: *Never use /remote in a pathname; it won't be available on an end-user machine.*

Hints on Error Checking and Other Advice

Here are some common problems and advice on how to avoid them:

- ◆ **Check all function return values.** Make sure that a function never exits the program if an error results, and that returned pointer values are not NULL.
- ◆ **Use printf() to facilitate debugging.** The `printf()` statements in your program statements are displayed in the Debugger Terminal window when you run the application in Debugger mode. The `printf()` statements are dropped when you run in stand-alone mode.

Note: *Once you have generated a clean `cdrom.image` file, eliminate the `printf()` statements for a slightly more efficient title.*

- ◆ **Read an integral number of blocks.** When reading files from the CD-ROM, you must read an integral number of disc blocks. For example, if `CMD_STATUS` reports that the blocksize is 2048, and the file size is 3277 bytes, you *must* read 4096 bytes.
- ◆ **Don't forget to recompile and retest.** If you changed your application don't forget to recompile it and test it once more.

Creating and Testing the Image File

This chapter provides information about creating and testing the CD-ROM image file you will use to master your CD-ROM disc.

This chapter covers the following topics:

Topic	Page
Background and Overview	10
Required Files and Folders	12
Preparing Simple cdrom.image Files	13
Preparing an Optimized CD-ROM Image	16
Testing the CD-ROM Image File on a Macintosh	20
Layout Tool Troubleshooting	20

Note: *The procedure described in this chapter produces good results for most titles. The actual improvement in file access time and startup time caused by the optimized layout tool and the Catapult feature depends on the title. In addition, a custom layout done by the developer may be faster than an optimized layout done by the layout tool.*

Background and Overview

This section is an overview of how to create a *cdrom.image* file of your title using the layout tool optimized for clustering and the Catapult file for rapid startup.

This section discusses:

- ◆ Layout tool background information
- ◆ Process for using the layout tool

Note: *Feel free to skip the background information first if you're mostly interested in the process.*

Layout Tool Background Information

The layout tool, which is provided on the 3DO M2 CD-ROM, takes all files in a folder and creates one CD-ROM image file. In the process, it places one or more avatars (instances) of the root, label, and directory in the image.

The current version of the layout tool allows optimized layout. First you collect information about your title using a simple image, then the layout tool uses that information for layout. The current layout tool also contains the Catapult feature which can significantly reduce the startup time of your CD-ROM.

Nonoptimized Layout

The original layout uses a “first seen, first placed” algorithm. Files are “seen” in alphabetical order within each directory; subdirectories are seen and placed recursively as they’re encountered.

This algorithm leads to delays during certain stages of a title’s startup. For example, a program’s sound (*.aiff*) files frequently require the loading of a DSP instrument (*.dsp*) template file. Because the *.aiff* and *.dsp* files are stored in different directories, they’re a fair distance apart on the CD-ROM, and the drive spends a lot of time moving the read head back and forth between different areas of the disc.

Layout Using the Cluster Optimizer

The current layout tool with its cluster optimizer locates files in clusters according to when they are used, not according to their position in the file system. If files from different directories are accessed one right after the other, they usually end up in adjacent sections of the CD-ROM. This greatly reduces head-seek time. Since files that are accessed in sequential time order are often laid out in sequential physical position, several files can be read in sequence with only one head seek.

The optimizer also places additional avatars (copies) of frequently-accessed files in several different locations on the disc. Small, frequently used files may be replicated up to 7 times. Large or infrequently used files may not be replicated at all.

The amount of avatar duplication depends on the amount of otherwise unused space in your file system. If your title has so much data that it almost fills a 640 MB CD-ROM, very little space is available for cluster optimization and avatar duplication. If your application uses only a few hundred megabytes, there is plenty of room for avatar duplication. This can improve performance. It also enables your application to run even if one or more avatar of critical files is not available because of scratching or other damage to the CD-ROM.

The clustering optimizer has some significant limitations. It is fairly common for an application to read pieces out of several different files in alternation. For example, when opening an audio *.aiff* file, the audio folio usually reads the first block of the *.aiff* file, then opens and reads the contents of a *.dsp* instrument file the *.aiff* file points to, and then reads the remainder of the *.aiff* file. The cluster optimizer can reduce the cost of the head seeks that take place when the drive seeks back and forth between the *.aiff* and *.dsp* files, but it cannot eliminate them. A similar problem arises whenever a program reads data from two or more files in parallel.

A further limitation of the clustering optimizer involves directories. Directories cannot actually be placed within clusters, because the size of a directory cannot be determined accurately until the placement (and number of avatars) of every file within that directory have been finalized. During a cluster optimization, directories are placed in small gaps between clusters. Accessing a directory at runtime (if the directory's contents are not currently in the directory cache) usually requires two head-seeks: one to the directory and one to the file that has just been opened. Clustering can reduce the cost of these directory-related seeks, but cannot eliminate them.

Catapult

Catapult is a feature of the layout tool. With the Catapult feature, the layout tool collects the CD-ROM data that your application needs at startup time (programs, data files, folios, directories, etc.) into a single area of the disc, in exactly the order needed. As a result, the startup process requires very few head-seeks. Proper Catapult optimization of your title can reduce its startup time significantly.

Catapult's operation takes place at the file system/driver interface level. Since you can use Catapult without making any changes to your programs or data, you should prepare one optimized image using Catapult, one optimized image without Catapult, and compare the two. The actual speedup varies greatly depending on the title itself.

Optimized Layout Process Overview

To use the clustering optimizer, you must perform the layout process three times and perform testing and collect access information in between. This results in the following steps, which are described in the remainder of this chapter:

- ◆ Perform your first layout process which results in a simple, nonoptimized *cdrom.image* file.
- ◆ Test your title.
- ◆ Create the base image for optimized layout. Use this image to exercise your title and collect access information using the 3DO Debugger's ability to create a *CD_Access.log* file which records all of the reads issued to the *cdrom.image* file during testing.
- ◆ Perform a second layout process during which the layout tool uses information from the *CD_Access.log* file to group files into clusters based on the actual patterns of use. This optimized *cdrom.image* file can then be burned onto a CD-ROM disc and tested from there.

Required Files and Folders

Before you can run the layout script file to create a CD-ROM image file, you must set up the files and folders for your title. You need to complete the steps discussed in detail in the following sections:

- ◆ Preparing Your Title's Files for the Layout Process, that is, placing them into a folder that contains a runtime version of the operating system and renaming them so that the layout tool can recognize them.
- ◆ Modifying the BannerScreen File

Preparing Your Title's Files for the Layout Process

Before you can use the layout tool to create the CD-ROM image, put all files needed in the image into one folder as follows:

1. Find your most recent operating system CD-ROM and locate the */cdrommaster* folder.

This folder contains a number of important system files.
2. Copy the */cdrommaster* folder to your hard drive.
3. Place copies of your application's executable and data folders (or folder hierarchies) in the */takeme* folder inside the folder named after the current operating system version.
4. Rename the main executable of the application to *launchme.m2*.

Modifying the BannerScreen File

The banner screen is a video image that the system displays early in the boot sequence. This video image is title-specific and is displayed while the system completes the boot sequence.

Banner Screen Requirements

The banner screen must meet certain requirements.

Note: *The following information is still being finalized. Please refer to your MEI representative for complete and final details of the banner screen.*

- ◆ The banner screen must be half-screen height (640 x 240) and have a color depth of 16 bits.
- ◆ The banner screen must contain the name of the title, prominently displayed. The company name alone is not sufficient.
- ◆ The banner screen must be specific to each title.
- ◆ The banner screen must contain appropriate copyright notices for the title.

Preparing Simple cdrom.image Files

The first step in creating your CD-ROM is to generate a simple, nonoptimized *cdrom.image* file. You create the CD-ROM image file using the *laytool* program. This program uses a *cdrom.tcl* script file to create the image file. This section describes:

- ◆ Editing the *cdrom.tcl* File for the Simple Image
- ◆ Creating the Simple Image

All files and folders needed to create the image file are in the */cdrommaster* folder named after the current version of the operating system.

To step through this section, you should have already set up your files, as explained in “Preparing Your Title’s Files for the Layout Process” on page 12.

About the *cdrom.tcl* File

The *cdrom.tcl* script file contains variables that affect the final form of the CD-ROM image file. Variables are defined with *set* commands in the script. If a command is preceded by an initial hash sign (*#*), it is considered a comment and won’t execute.

Editing the `cdrom.tcl` File for the Simple Image

To edit the `cdrom.tcl` file for first-time layout, follow these steps:

1. Launch MPW.
2. From the Directory menu, choose Set Directory and select the folder inside the `cdrommaster` folder that is named like the most recent operating system.
3. From the File menu, choose Open and select the `cdrom.tcl` file.

Note: *The `cdrom.tcl` file is commented extensively; if you pay attention to the comments, you should find editing the file quite easy.*

4. Edit the file so the variables have the values shown in Table 3-1:

Table 3-1 Variable settings for simple image creation.

Variable	Setting
<code>imagefile</code>	Pathname of a <code>cdrom.image</code> file to be created on a hard disk that has at least 640 MB of available space.
<code>label</code>	Set to "cd-rom" for initial testing of the CD-ROM image.
<code>takedirectory</code>	Pathname of the <code>/takeme</code> folder that contains the title.
<code>megabytes</code>	Rule of Thumb: Use 110% of the size of the <code>/takeme</code> folder (with your executable and data files), and use that number as the number of megabytes. If you allocate too little, the layout process terminates. If you allocate too much, the process takes longer but there won't be other problems. The number of blocks must be a multiple of 16. The number of kilobytes must be a multiple of 32.
<code>directoryavatars</code>	The number of directory avatars defaults to 3 and may be safely reduced to 2.
<code>rootavatars</code>	The number of root avatars defaults to 7 and may be safely reduced to 2.
<code>labelavatars</code>	The number of label avatars defaults to 2, and must not be changed.

The different avatars are discussed in some detail in the `cdrom.tcl` file itself.

Creating the Simple Image

To create a CD-ROM image using the layout tool, follow these steps:

1. In the MPW Worksheet window, make sure `/cdrommaster/_version_` is the working directory.

The `/cdrommaster/_version_` folder contains the `/takeme` folder with your executables and data.
2. Type `laytool < cdrom.tcl` and press the Enter key (not the Return key).

After you've started the image creation process, the layout tool does the following:

- ◆ Assembles the Macintosh file hierarchy of your title into a Portfolio file system.
- ◆ Creates the `cdrom.image` file.
- ◆ Creates the `filemap.out` file that is needed for creating an optimized CD-ROM image.

At the end of the process, the MPW shell should display the line "layout successful."

If you don't see the line "layout successful," or if something else goes wrong:

- ◆ Look at "Layout Tool Troubleshooting" on page 20 below.
- ◆ If you can't solve your problems with that information, select the status messages the layout tool printed and save them to a file for use by MEI Developer Technical Support.

Testing the Simple Image

To test the CD-ROM image file, follow these steps:

1. Copy the `cdrom.image` file into your release folder, where your debugger resides.
2. Temporarily move the remote folder out of the release folder
3. In the debugger Target menu, select Hardware Reset.
4. The shell prompt should read `/cd-rom`
5. Type `launchme.m2`

The Debugger runs your title from the image file and you can perform the first round of testing.

Preparing an Optimized CD-ROM Image

Preparing an optimized CD-ROM image starts with creating a basic image, described in “Preparing Simple cdrom.image Files” on page 13. The following sections discuss the additional steps needed to produce an optimized image, which include:

- ◆ Creating the Base Image for Information Collection
- ◆ Preparing the Debugger for Information Collection
- ◆ Running Your Title to Collect Access Information
- ◆ Running the Layout Tool to Create an Optimized Image

Creating the Base Image for Information Collection

The base image for information collection is identical to the simple image discussed in “Creating the Simple Image” on page 15.

Preparing the Debugger for Information Collection

The optimizing layout tool requires information about when and how often the different files your title consists of are accessed when someone uses the title.

To set up the Debugger to collect file access information, follow these steps:

1. Make sure that you have run the FlashRomTool to load the file *DEVROM.m2.flash.unenc.str* into your Flash ROM, as described in the *3DO M2 Debugger Programmer's Guide*.
2. Make sure the 3DO Debugger is not running.
3. Put the debugger Preferences file *3DODebug.Prefs* into the trash.
4. Make sure there is no CD in the 3DO M2 external CD-ROM drive.
5. Make sure that there is only one *cdrom.image* file in your release folder. Temporarily move the remote folder out of your release folder.
6. Launch the 3DO Debugger.
7. In the Target Setup window, click OK.
8. When prompted for a script to execute, select *debugger.flash.scr*.
9. The system should boot for the shell prompt, which should be */cd-rom>*
10. From the Execution menu, select Stop.
11. From the Tools menu, select CD Access Log. Save the file *CD-Access.Log* into your */cdrommaster/_version_folder*.
12. From the Target menu, select HW Reset.
13. The system should again boot to the shell prompt */cd-rom>*

14. Type `launchme.m2`.

Running Your Title to Collect Access Information

You are now set up for collecting the file access information required by the optimized layout tool. This section steps through running your title from the CD-ROM image file to collect the information.

Note: *The person collecting file access information should be thoroughly familiar with the title.*

With the 3DO Debugger running, follow these steps:

1. From the File menu, choose Special Mode.
This puts the Debugger into special mode to make CD-ROM emulation as fast as possible.
2. With all settings for the Debugger in place, go through your title as follows:
 - ◆ Go through the title's startup sequence.
 - ◆ At each major user-interaction point, for example, each time a menu lets the user select a game level, pause for a second or so before making a selection.
 - ◆ Enter each major branch of the title only once if possible, so that file access information for that branch can be recorded.

The goal is to "exercise" the set of files used by each level, so that the optimizer can observe which files are used together.
3. After "touching" each level or branch in the title, press the Stop button on the control pad to exit the program.
4. Click the Macintosh mouse to terminate Debugger special mode.
5. From the Tools menu, select CD Access Log.
6. Quit the Debugger.

Running the Layout Tool to Create an Optimized Image

To create an optimized CD-ROM image file, you have to make the CD access log available to the `cdrom.tcl` script file that you run to create a CD-ROM image.

This section shows how to set up all required files and how to run the layout tool again to create the optimized image.

Editing the `cdrom.tcl` File for the Optimized Image

To edit the `cdrom.tcl` file for the optimized image, use the settings in Table 3-2, which differ from the settings for the nonoptimized image you used earlier:

Table 3-2 Variable settings for optimized image creation.

Variable	Setting
label	cd-rom (the default). Make sure the label is NOT set to remote at this stage or you won't be able to debug the optimized image.
megabytes	Number of megabytes of CD-ROM space your software will occupy. Larger is better; the more storage space you allocate (in excess of the amount actually needed for one copy of each file), the more room the optimizer and Catapult have to make multiple avatars of your title's files, and the more effective the optimization and clustering will be.
catapult-megabytes	<p>Set catapultmegabytes to the number of megabytes of CD-ROM space that you wish to make available for the Catapult file. If your application already uses most of the space on the CD-ROM, you can make this a fairly small number. If your application has lots of room to spare, request a larger Catapult file (20 MB or more).</p> <p>Since the actual benefit of Catapult varies greatly depending on the title, consider making several images, one of them with catapultmegabytes commented out.</p>

If you used Catapult, you can exclude certain files or entire directories, from being included in the Catapult file (see "Catapult" on page 11). This can make sense for Cinepak or other full-motion-video files: they take up a lot of space in the Catapult file and usually don't benefit much from being catapulted. To do this, include a statement using one of the following formats in your `cdrom.tcl` file:

```
excludecatapult "foo/bar/baz.cinepak"  
excludecatapult "foo/bar/"
```

The `excludecatapult` command accepts a single parameter, the Portfolio pathname to a file or directory. In either case, the path is relative to the root directory of the file system being created. The first format identifies a specific file which is to be excluded from the Catapult file. The second format (with a trailing `"/"`) identifies a directory, and specifies that all files found in that directory are to be excluded from the Catapult file.

Dealing with Large Files

If you have a large Cinepak file, or any other large file which is “streamed” into memory at high speed, you should do one of two things:

- ◆ *Either* exclude the file from the Catapult file using the `excludecatapult` command
- ◆ *or* make sure that the entire file is included in the Catapult file.

You need to make sure you don’t have half of a file in the Catapult file and half of it excluded. This can occur if during your test-and-exercise run you play halfway through the file and then use the control pad to skip the rest of the introductory sequence. In this case, there may be an extra seek when the catapulted CD-ROM image accesses the file. It reads the first portion of the data from the Catapult file, but is forced to find an avatar of the file itself to get the remainder.

Starting the Image Creation Process

After you’ve run the title with CD Access Log on, you should find a *filemap.out* file and a *CD_Access.log* file in the */cdrommaster/_version* folder.

1. Rename the *filemap.out* file to *filemap.in*.
2. Make sure the *CD_Access.log* file resides in the */cdrommaster/_version* folder.
3. Delete (or rename and move) the old copy of the *cdrom.image* file in the */cdrommaster/_version* folder.
4. Run the layout tool program again, using the newly edited script. The *CD_Access.log* and *filemap.in* files are automatically used as input.

The tool goes through several stages of file access analysis: clustering, file organization, cluster placement, and avatar assignment. It then writes a new *cdrom.image* file, which is optimized for the access performed while testing the base *cdrom.image* file.

Testing the CD-ROM Image File on a Macintosh

After creating the CD-ROM image file, test it on the Macintosh to make sure that all necessary files are included in the image.

To test the CD-ROM image file, follow the steps described in “Testing the Simple Image” on page 15:

Layout Tool Troubleshooting

This section explains some of the error messages or problems you can encounter while running the layout tool and provides possible solutions.

Error Message: Layout failed; unable to allocate avatar(s)

The placement of avatars can result in insufficient contiguous space for a large file and result in following message:

```
layout failed: unable to allocate avatar(s)
for:MyReallyBigMovieFile
```

If that message appears, increase the total amount of space assigned to the image file or reduce the number of avatars.

Error Message: CD Image creation error: “Could not assign avatar(s) for launchme.m2”

This message indicates the layout tool could not write all the avatars because the *cdrom.tcl* file did not allocate enough space for the image. Try the following:

1. In the *cdrom.tcl* file, increase the amount of space allocated for the *cdrom.image* file by editing the following line:

```
set megabytes n
```

The recommended formula to compute the number of megabytes, discussed earlier, is the following:

```
((size of takeme folder) * 1.1)
```

2. Decrease the number of directory avatars or root avatars in the *cdrom.tcl* file.

Error Message: CD Image creation error: “Unable to match glob patterns”

There is an empty folder in the directory you created; remove it.

Problem: Title does not run

To track down the problem, ask yourself the following questions:

- ◆ Are there hard-coded pathnames using */remote* in your program? Your program must use relative pathnames.

Make sure that you use the version of the layout tool distributed with the most recent release of 3DO software.

- ◆ When your application reads files from the CD-ROM, it must read an integral number of disc blocks. For example, if `Cmd_Status` reports that the blocksize is 2048 and the file size is 3277 bytes, the program *must* read 4096 bytes.

Message: "O/S error -34 allocating space"

This error indicates that there is not enough contiguous space on the hard drive to lay down the CD-ROM image. Remember you need the amount of space you have defined in the `cdrom.tcl` file. You can also edit the `cdrom.tcl` file so the layout tool places the `cdrom.image` file on a hard disk that has enough space.

Note: *Consider defragmenting your hard disk if you have this problem.*

Mastering and Testing the CD-ROM

This chapter explains how to create a master CD-ROM on a CD-ROM recording station and perform the necessary testing. The chapter includes the following topics:

Topic	Page
Preparation and Setup for the Mastering Process	24
Creating a CD-ROM Disc for Testing	24
Testing the CD-ROM Disc Using the Debugger	28
Creating a Master CD-ROM Disc	29

For More Information

You can find additional information that is relevant to the CD recording process in the following chapters:

- ◆ Recording station setup and recommended recorders and recording media are discussed in Chapter 1, "Hardware and Software Requirements."
- ◆ Chapter 5, "Delivery for Authorization," provides summary information of all currently known requirements.
- ◆ Appendix A, "CD-ROM Mastering Checklist," is helpful during the mastering process.

Preparation and Setup for the Mastering Process

Before you can begin the actual mastering process, you must prepare your hardware and software.

This section discusses:

- ◆ Maximizing Recording Speed
- ◆ Setting Up the QuickTOPIX Software

Maximizing Recording Speed

Because the compact disc recorder has very little buffering, your Macintosh must maintain a continuous 300 KB/s data stream to it during recording. Therefore, prepare your Macintosh as follows:

1. Disconnect the Macintosh from any networks and shut down all network drivers.
2. Remove system extensions that attempt to monitor SCSI bus activity, such as those that implement flashing menu bar icons to show disk activity.
3. Keep the compact disc recorder in a stable environment, in particular:
 - ◆ Make sure there is no movement near the recorder; it can cause a faulty CD-ROM disc.
 - ◆ Don't place the Macintosh hard drive on top of the recorder or near loudspeakers.

Setting Up the QuickTOPIX Software

The QuickTOPIX software lets you master a CD-ROM disc in several different formats, two of which you use during the mastering process:

- ◆ Choosing Block Image creates a disc in 3DO format for testing.
- ◆ Choosing Apple HFS image creates an HFS disc for delivery to MEI for authorization.

This section discusses setting up the software for both image formats and lists the files created during that process.

Note: *Setting up the software for both image formats before starting the mastering process saves time and trouble later.*

Setup for Mastering a CD-ROM for Testing

To set up the software for mastering a CD-ROM disc for testing, follow these steps:

1. Make sure that you have installed the latest version of the QuickTOPIx software, including the QuickTOPIx system extension.
2. Find the QuickTOPIx Chooser folder in the QuickTOPIx folder on your Macintosh.
3. Double-click on the Block Image icon in the QuickTOPIx Chooser folder.

The setup screen for recording a CD-ROM image file to disc appears. This type of recording creates a disc for testing.

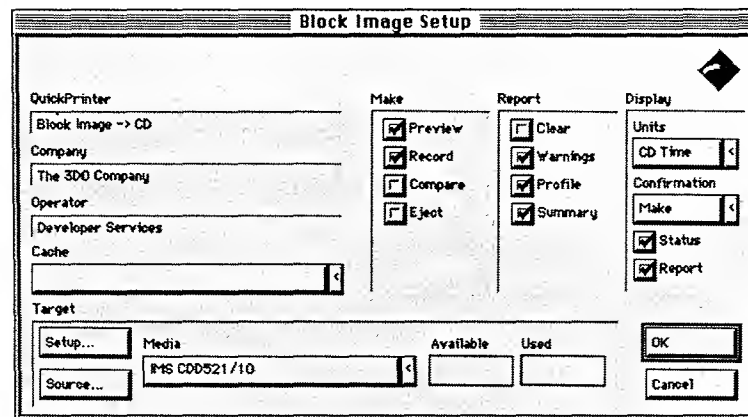


Figure 4-1 QuickTOPIx setup screen for block image format.

4. Type in the company name and operator name, deselect the Preview check box in the Make section, and click OK.
5. In the dialog box that appears, click Yes to save your changes.

Setup for Mastering a CD-ROM

To set up the software for mastering an HFS image, follow these steps:

6. Double-click on the Apple HFS icon.

The Apple HFS setup screen for mastering a disc of an entire Macintosh volume appears. This type of mastering process creates a disc that you send to MEI for authorization.

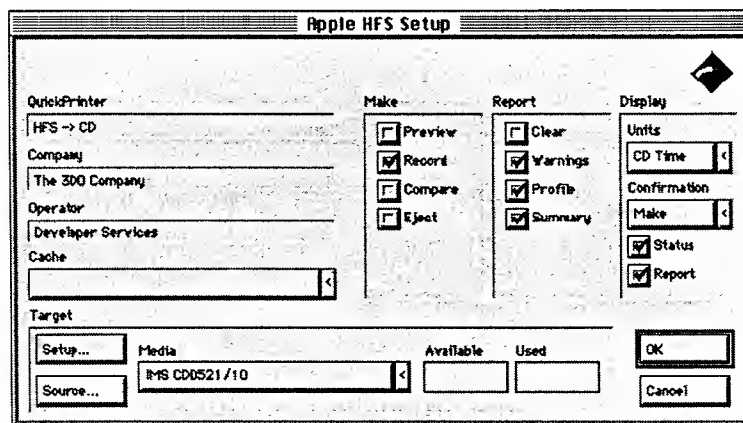


Figure 4-2 QuickTOPIX setup screen for Apple HFS format images.

7. Type in the company name and operator name, deselect the Preview check box in the Make section, and click OK.
8. In the dialog box that appears, click Yes to save your changes.

Files Created by the Software Setup

After completion of the software setup for Block image format and Apple HFS image format, there are four new files on your desktop:

- ◆ *Apple HFS* and *Block Image* files contain the information you just entered.
- ◆ *Apple HFS.r* and *Block Image.r* are log files that the QuickTOPIX system maintains and updates whenever you use one of the two images.

Creating a CD-ROM Disc for Testing

This section discusses the first step in the actual mastering process: Creating a CD-ROM in Block Image format for testing.

To master the CD-ROM disc, follow these steps:

1. Place the CD-ROM in the CD recorder's drive and push the close button. Two lights should be on, indicating that the power is on and the disc is active. Wait for the Disc In light to stop blinking.
2. Click on the *cdrom.image* file that you want to record to disc and drag it on top of the Block Image icon on your desktop.

The window shown in Figure 4-3 appears:

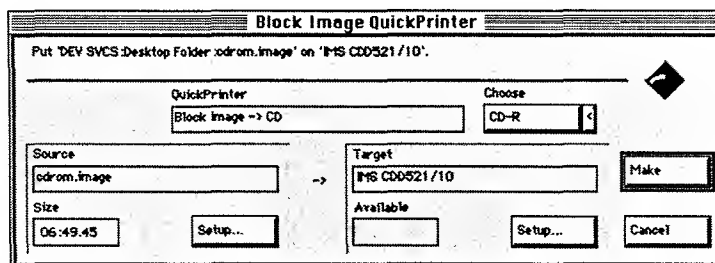


Figure 4-3 QuickTOPIX dialog for creating a block image creation.

3. Click the Make button.

The *cdrom.image* file is recorded to disc.

The QuickTOPIX status window appears and reports the progress of the mastering process. The transfer rate should be between 295 KB/s and 300 KB/s. When the process is finished, "Ready" appears in the status window.

Caution: On most recorders, the drawer opens when the process is complete. Do not open the drawer too soon yourself. The Writing light turns off once, then turns on as the lead data is written.

4. Quit QuickTOPIX.
5. Remove the new master disc, label it with date and contents, and mark it as 3DO formatted.

You are now ready to test your newly mastered CD-ROM.

Testing the CD-ROM Disc Using the Debugger

Testing the CD-ROM involves the following two steps:

- ◆ Setting up the System for Testing a CD-ROM
- ◆ Running the CD-ROM

Setting up the System for Testing a CD-ROM

To run your title from a CD-ROM using the Debugger, do the following to set up your environment:

1. Delete or rename any *cdrom.image* files on your Macintosh.
2. Temporarily move the remote folder out of your release folder.

Note: If you test your CD-ROM from the Debugger, `printf()` statements in the program are displayed in the Terminal window on the Macintosh. This helps you debug your title.

Running the CD-ROM

Follow the steps below to run your title from the CD-ROM disc:

1. Insert the Master CD-ROM into the CD-ROM drive attached to the 3DO M2 Development Card.
2. Launch the 3DO Debugger on the Macintosh (or select Hardware Reset from the Execution menu if the Debugger is already running).
3. The shell prompt should read `/cd-rom>`
4. Type `launchme.m2`.

Debugging Hints

This section discusses what you can do if you have problems running your title from the newly mastered CD-ROM disc:

- ◆ Stepping Through the Title on the CD-ROM Disc
- ◆ Pulling Files Off the CD-ROM

Stepping Through the Title on the CD-ROM Disc

If a Symbols file for the title is still available in the */remote* directory, you can step through the title on your CD-ROM disc using the Debugger.

The *3DO M2 Debugger Programmer's Guide* describes this process in more detail.

Pulling Files Off the CD-ROM

If you are unsure which version of your executable you have placed on the disc, or if you need to pull files off the CD-ROM for some other reason, you can use the normal shell `copy` command. The copy utility does not accept filenames with spaces.

You can pull individual files from the CD-ROM disc and place them on your Macintosh for inspection by typing the following at the Terminal window:

```
copy <source> <destination>
```

For example, the following command will copy the file *afile* from */cd-rom* to */remote*.

```
/remote> copy /cd-rom/afile /remote
```

Note: The source filename has a leading slash (/); the destination filename does not unless you specify a file inside */remote*

Creating a Master CD-ROM Disc

When you're satisfied that your title on the CD-ROM disc is completely functional in stand-alone mode, you can create the HFS-formatted discs to send to MEI Developer Services for authorization.

To create the final master CD-ROM in HFS format, follow these steps:

1. Place the CD-ROM in the disc recorder's drive and push the close button. Two lights should be on, indicating that the power is on and the disk is active. Wait for the Disc In light to stop blinking.
2. Click on the partition of your hard disk that you want to record to disc and drag it on top of the Apple HFS icon on your desktop.

The dialog shown in Figure 4-4 appears:

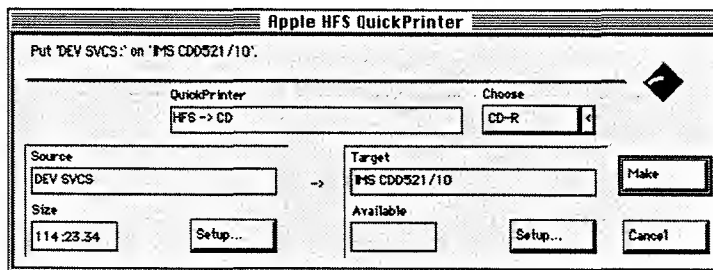


Figure 4-4 QuickTOPIX dialog for creating a CD-ROM for delivery.

3. Click on the Make button.

The partition is recorded to disc.

The QuickTOPIX status window appears and reports the progress of the mastering process. The transfer rate should be between 295 KB/s and 300 KB/s. When the process is finished, "Ready" appears in the status window.

Caution: *Do not open the drawer too soon. The Writing light turns off once, then turns on as the lead data is written.*

4. Remove the new master disc, label it with date and contents, and mark it HFS formatted.
5. Test the master disc on your 3DO M2 Station before sending it off to MEI for authorization.

Delivery for Authorization

This chapter summarizes the authorization requirements and process, briefly discussing the following topics:

- ◆ CD packaging and artwork requirements
- ◆ Files required for authorization

Note: *More details are available from your MEI account manager. This section is intended to give you an overview of the authorization requirements.*

About the Authorization Requirements

When you submit a title for authorization, MEI checks that all requirements are met. Table 5-1 summarizes the requirements, providing references to more detailed information as appropriate.

Warning: MEI may not authorize titles that don't meet the requirements outlined here. Contact your MEI representative for authorization requirements.

Table 5-1 *Requirements for Authorization.*

Requirement	Discussion
CD packaging and artwork	CD artwork and CD packaging artwork are inspected for adherence to MEI guidelines.
Files you have to deliver	<p>MEI needs to see the <code>/takeme</code> directory of your title to verify that you created your <code>cdrom.image</code> file with the appropriate system files.</p> <p>When you submit your title for authorization, include:</p> <ul style="list-style-type: none">◆ the <code>cdrom.image</code> file◆ the <code>/takeme</code> directory (without executable and data files). <p>Deliver these files in HFS format on CD-ROM. Send at least two copies. See the Disc Processing Form for detail; a copy of that form is required for authorization.</p>

For More Information

Table 5-1 only provides an overview of authorization requirements. For more detailed information, contact your MEI representative.

CD-ROM Mastering Checklist

Reading List

- ☐ Before you start the mastering process, read the previous chapters in this document. This list summarizes the information.
- ☐ Look at Appendix B, “CD-ROM Mastering Etiquette,” for additional useful information about mastering a 3DO M2 CD-ROM disc.

Preparing for the Mastering Process

- ☐ Consider the application size limitations (see “Application Size Limitations” on page 6).
- ☐ Make sure the application doesn’t use absolute pathnames. Use relative pathnames.
- ☐ Check all function return values. Make sure that a function never exits the program if an error results and that returned pointer values are not NULL.
- ☐ To facilitate debugging the CD-ROM image, use `printf()` statements liberally throughout the program.
- ☐ When reading files using block I/O from the CD-ROM, the program *must* read an integral number of disc blocks. For example, if `CmdStatus` reports that the blocksize is 2048, and the file size is 3277 bytes, you *must* read 4096 bytes.
- ☐ Recompile and test your application.
- ☐ Make a copy of the folder named like the most recent version of the operating system that is located in the `/cdrommaster` folder and place it on the desktop.
- ☐ Place your application’s executable and data files into the `/takeme` folder in the `/cdrommaster` folder you just placed on your desktop.

- ☐ Rename the executable of the application to *launchme.m2*. File names are *not* case sensitive.

Creating and Testing a Simple CD-ROM Image File

- ☐ Launch MPW.
- ☐ From the Directory menu, choose Set Directory, then select the */cdrommaster/_version_* folder.
- ☐ Open the *cdrom.tcl* script file and uncomment and edit the variables as described in "Editing the *cdrom.tcl* File for the Simple Image" on page 14.
- ☐ Save the *cdrom.tcl* file.
- ☐ In the MPW Worksheet window, type `laytool < cdrom.tcl` and press the Enter key to run the script.
- ☐ Make sure a "layout successful" message appears at the end of the process.

A *cdrom.image* file now resides in the */cdrommaster/_version_* folder.

If an error occurs, see "Layout Tool Troubleshooting" on page 20 or call MEI Developer Technical Support.

- ☐ Start the system and type into the Terminal window:
`launchme.m2`

The Debugger runs your title from the CD-ROM image file, and you can perform the first round of testing.

Creating an Optimized Image

The optimizing layout tool requires information about when and how often the different files your title consists of are accessed when someone uses the title.

Follow the steps below to create an optimized image:

- ☐ Make sure that you have run the FlashRom Tool to load the file *DEVROM.m2.flash.unenc.str* into your Flash ROM, as described in the *3DO Debugger Programmer's Guide*.
- ☐ Make sure the 3DO Debugger is not running.
- ☐ Put the debugger Preferences file *3DODbg.Prefs* into the trash.
- ☐ Make sure there is no CD in the 3DO M2 external CD-ROM drive.
- ☐ Make sure that there is only one *cdrom.image* file in your release folder. Temporarily move the remote folder out of your release folder.

-
- ☐ Launch the 3DO Debugger.
 - ☐ In the Target Setup window, click OK.
 - ☐ When prompted for a script to execute, select *debugger.flash.scr*.
 - ☐ The system should boot for the shell prompt, which should be */cd-rom>*
 - ☐ From the Execution menu, select Stop.
 - ☐ From the Tools menu, select CD Access Log. Save the file *CD-Access.Log* into your */cdrommaster/_version_* folder.
 - ☐ From the Target menu, select HW Reset.
 - ☐ The system should again boot to the shell prompt */cd-rom>*
 - ☐ Type *launchme.m2*.
 - ☐ From the File menu, choose Special Mode. The application should start running since the label was set to remote.
 - ☐ Go through the title as follows:
 - ◆ Go through the startup sequence.
 - ◆ At each major user-interface interaction point—for example, each time a menu lets the user select a game level—pause for a second or so before making a selection.
 - ◆ Enter each major branch of the title once—if possible, exactly once each—and let the level perform its typical file accesses.
 - ☐ When you're finished, press the Stop button on the control pad to exit the program, then click the Macintosh mouse to terminate Special Mode.
 - ☐ From the Tools menu, select CD Access Log, then quit the Debugger.
 - ☐ Rename the *filemap.out* file in the */cdrommaster/_version_* folder to *filemap.in* and make sure the *CD_Access.log* file resides in the */cdrommaster/_version_* folder.
 - ☐ Edit the *cdrom.tcl* file a second time using the values described in "Editing the *cdrom.tcl* File for the Optimized Image" on page 18.
 - ☐ Save the *cdrom.tcl* file.
 - ☐ Delete the existing *cdrom.image* file and run the layout tool again. The tool uses the *CD_Access.log* file and the *filemap.in* file to create the optimized *cdrom.image* file.

Testing the Optimized CD-ROM Image on the Macintosh

- ☐ Make sure that there is only one *cdrom.image* file on your Macintosh.
- ☐ Rename all source files (executables and data files) on your Macintosh to make sure that they are not used by the Debugger during testing.

Mastering the CD-ROM Disc

- ☐ Connect the hard drive with the *cdrom.image* file to the compact disc recorder.
- ☐ Copy the *cdrom.image* file into a clean partition on the recording station Macintosh.
- ☐ Disconnect the Macintosh from any networks, shut down network drivers, and remove any system extensions that attempt to monitor SCSI bus activity, such as those that use flashing menu bar icons to show disk activity.
- ☐ Make sure that the compact disc recorder is isolated from vibrations.
- ☐ Make sure that you have installed the latest version of the QuickTOPIX software, including the QuickTOPIX system extension.
- ☐ Double-click on the Block Image icon in the QuickTOPIX Chooser folder.
- ☐ Enter your company's name and the operator name, de-select the Preview check-box in the Make section, and click OK, then click Yes to save your changes.
- ☐ Find the QuickTOPIX Chooser folder in the QuickTOPIX folder and open the Apple HFS application.
- ☐ Enter your Company name and Operator Name, de-select the Preview check-box in the Make section, and click OK, then click Yes to save your changes.

There are now four new files on your desktop:

 - ◆ *Apple HFS* and *Block Image* contain the information you just entered.
 - ◆ *Apple HFS.r* and *Block Image.r* are log files that the QuickTOPIX system maintains and updates whenever you use one of the two images
- ☐ Place the CD-ROM in the drive and press the close button. Two lights should be on, indicating that the power is on and the disc is active. Wait for the Disc In light to stop blinking.
- ☐ Drag the icon that represents the *cdrom.image* file you want to record on top of the Block Image icon on your desktop.
- ☐ In the dialog that appears, click the Make button.

-
- ☐ When the write process is finished, "Ready" appears in the Status window. The Writing light turns off, then turns on again as the lead data is written.
 - ☐ Quit QuickTOPIX and remove the new master disc for testing.

Testing the CD-ROM

Start by testing the CD-ROM on a 3DO M2 development station.

Preparing Materials for Authorization

The CD-ROM disc you send to MEI Developer Services for authorization must be in HFS format.

- ☐ Place the CD-ROM in the compact disc recorder's drive and wait for the Disc In light to stop blinking.
- ☐ Click on the *cdrom.image* file and drag it on top of the Apple HFS icon on your desktop.
- ☐ In the window that appears, click the Make button.
- ☐ When the write process is finished, "Ready" appears in the Status window. The Writing light turns off, then turns on again as the lead data is written.
- ☐ Quit QuickTOPIX and remove the new master disc.

CD-ROM Mastering Etiquette

A 3DO M2 title uses the CD-ROM as its data storage medium. CD-ROM technology and its use in multimedia applications is still in its infancy. As The 3DO Company and its developers have worked with the CD-ROM device over the past year, they have learned a lot about maximizing drive performance and about the capabilities and limits of the medium.

This appendix presents a collection of this information. It is intended to help you understand the issues involved and provide guidelines for creating titles for the CD-ROM drive.

The appendix discusses the following topics:

Topic	Page
CD-ROM Basics	40
Programming Do's and Don'ts	42
Head Seeks and CD-ROM Mechanism Lifetime	44

3DO and its developers are forging new frontiers in CD-ROM technology, pushing the technology to its limits. With careful and well-planned execution, programmers can increase CD-ROM drive performance and reliability by employing several techniques for reducing head seeking, and making sure that they gracefully recover from data errors or slower-than-expected data delivery.

CD-ROM Basics

This section provides some background information, including the following topics:

- ◆ Constant Linear Velocity
- ◆ 3DO Drive Specifications
- ◆ Data Rate Issues
- ◆ Error Detection and Correction

Constant Linear Velocity

Unlike magnetic disk storage mediums that rotate with constant angular (or, rotational) velocity (CAV), the CD-ROM disc rotates at constant linear velocity (CLV).

Linear velocity is related to angular velocity as follows:

Linear velocity = angular speed * distance from center of disc
 $v = w * r$

In CLV, v is held constant. As r increases, w must decrease, and vice versa.

Double-speed CD-ROM drives must spin at roughly 1000 rpm at the center of the disc, and 400 rpm at the outer edge. Quad-speed CD-ROM drives must spin at approximately 2000 rpm at the center of the disk, and at 800 rpm at the outer edge.

3DO Drive Specifications

3DO CD-ROM hardware licensees must meet the following specifications when producing drives for 3DO:

Table A-1 3DO drive specifications.

Specification	2X Drive	4X Drive
Average sequential throughput	300 KB data/s	600 KB data/s
Buffer memory size	32 KB	32 KB
Average access time including latency (1/3 stroke)	500 ms	500 ms
Worst case access time including latency (full stroke)	800 ms	800 ms

Data Rate Issues

Some developers make the assumption that a double-speed drive will always allow 300 KB/s of data streaming off the disc, and that a quad-speed drive will consistently stream 600 KB/s of data off the disc.

In reality, however, you almost *never* get the theoretical continuous maximum flow of data (although you may come close). The following factors cause performance hits in data delivery:

Disc Seeks

Each time a title needs to seek for a file on a CD-ROM, there is a delay. Every seek (from one file or directory to another) costs at least 100 msec. Some seeks are up to 1000 msec.

Seek times on the disc in a CLV system are dependent on the distance that the read head must move, the amount of increase or decrease in rotational velocity that is required to read the data at the new position, and the drive mechanism itself. Be aware that since disc seeking profiles vary with the manufacturer of a given CD drive.

Rotational Latency

Once the head has seeked to a specific track, it needs to find the appropriate sector on that track. The disc needs to rotate to the sector. This latency may be as long as 150 milliseconds.

Error Detection and Correction

CD-ROM disc read errors can have many different causes. There can be dust, scratches, or fingerprints on the surface of the disc. There can also be bubbles or contaminants in the substrate of the disc. Errors might also occur when there are tracking or focusing problems.

Error detection and correction algorithms are complex and beyond the scope of this document. Suffice it to say that read error problems *can* have significant negative effects on the data transfer rate; however, the majority of errors are recoverable in a manner transparent to the developer's code.

Programming Do's and Don'ts

Clearly, CD-ROM is a somewhat fragile and imperfect storage medium. But there are things a programmer can do, and avoid doing, to reduce problems, increase data integrity, and increase data rates. This section provides some advice, looking at the following topics:

- ◆ Making Assumptions about Data Delivery Rates
- ◆ Handling Data Delays
- ◆ Alternating File Reads
- ◆ Testing Performance
- ◆ Other Things You Should Do

Making Assumptions about Data Delivery Rates

Don't assume a certain data delivery rate from the drive to a data buffer.

Do check the real error code in the IOReq structure before reading from a data buffer to make sure that good data is in the buffer:

```
err = DoIO(ioReqItem, &ioInfo);
if (err < 0 || (err = ioReq->io_Error) < 0)
{
    printf(Error getting status:");
    PrintfSysErr(err);
}
```

Handling Data Delays

Don't abort out of a task if that task can't read data from a data buffer, or if the task does not receive data as fast as it expects.

Do handle data delays gracefully. For example, an application should not crash if there is a data delay; it should be designed to minimize the problem and glitch in the video stream and continue when it receives the data it needs.

Alternating File Reads

Don't alternate reads, rapidly, between two or more files. This almost always requires seeking, which imposes a fairly drastic delay (see section above).

Do consider combining data from many small files (e.g., sound effects) into one file, with a compact table-of-contents at the beginning of the file. At the very least, if you must have two or more files, make sure the files are physically as close to each other on the disc as possible.

Testing Performance

Don't assume that the program's performance when run in development mode (from /remote) is the same as when it's run on a CD-ROM, even if you are running with CD Emulation in Special Mode in the Debugger.

Do test actual performance on a physical CD-ROM disc, in all currently-available 3DO players.

Other Things You Should Do

- ◆ Issue reads in fairly big chunks (32K or larger) wherever possible.
- ◆ When possible, use read-ahead buffers before branches to reduce branching seek delays.
- ◆ Provide adequate buffering to survive a one-revolution delay (up to 150 milliseconds).
- ◆ If files are used together (e.g., in one level of a game), put them in the same directory.
- ◆ If files are accessed in a strictly sequential manner, name them in ascending order.
- ◆ Use data-streaming techniques to combine multiple types of data into a single file which can be read sequentially. This eliminates seeks between files.
- ◆ Use the Layout Optimizer, which offers the following features:
 - ◆ Attempts to cluster files together on the disc in the same order that they're used during program testing.
 - ◆ Places one avatar of each file into cluster where it's used most frequently.
 - ◆ Attempts to place additional avatars of commonly-used files into other clusters which use the file.
 - ◆ Organizes clusters with most seeks nearest center to reduce latency.
 - ◆ Attempts to place directory avatars near files.
 - ◆ Is very effective at reducing the number of seeks and shortening those seeks which do occur.
 - ◆ Does not guarantee "zero seek" performance.
 - ◆ Greatly improves the titles initial load time by interleaving the files needed for booting (Catapult).

Head Seeks and CD-ROM Mechanism Lifetime

Many 3DO titles employ creatively authored “attract mode” sequences where the application, in the absence of user intervention, displays interesting or attractive sequences of audio and video. In general, this is a good thing. Some of these sequences, however, are designed in such a way that the CD-ROM drive has to do a great deal of seeking - causing a condition known as “disc thrashing.” The result is more sluggish performance, and much more wear and tear on the CD-ROM mechanisms than is necessary.

CD-ROM mechanism lifetime is limited by several factors. If the drive is not operating, then it will last a very long time. If it is in linear play (like a car cruising down the freeway), then it will still last a reasonably long time. If it is constantly seeking back and forth (like a car in stop and go traffic, constantly accelerating, and braking()) then its lifetime will be severely reduced.

Here, then, are strongly-recommended guidelines for attract mode screens:

- ◆ If possible, make attract mode sequences load once and run continuously without accessing the disc;
- ◆ If the application requires something fancier, then make the disc accesses linear. Typically, this would involve a sequence of canned video. 3DO provides many tools to help author such sequences, and to weave the audio and video into a single stream so that no seeks are required. Avoid non linear disc accesses;
- ◆ Make the best use of the latest layout tools. Call and verify the versions of the tools you have. The layout tools work to minimize seeks under all conditions;
- ◆ Make use of any or all of the other suggestions in the sections above;

Seek time is simply wasted time that wears out the hardware.

Optimization Issues

To minimize startup time and file access time, consider the following issues:

- ◆ A title that uses the optimized layout procedure described in this document usually has better file access time than a title that uses the simple layout only. An exception may be a title using custom layout fine-tuned by the developer.
- ◆ If you use the optimized layout described in this document, it is critical that the person collecting access information is thoroughly familiar with the title.
- ◆ Catapult is a relatively new feature to speed up startup time. Since the effect of Catapult depends on the title in question, consider preparing one image using Catapult, one without, and comparing the two.

Index

Numerics

3DO drive specifications CDM-40

A

Apple HFS file CDM-26
Apple HFS format CDM-25
Application size limitations CDM-6
avatars "could not assign" CDM-20

B

BannerScreen file CDM-13
 requirements CDM-13
Block Image file CDM-26
Block Image icon CDM-27

C

Catapult
 dealing with large files CDM-19
 overview CDM-11
catapultmegabytes variable CDM-18
CD Access Log CDM-19
CD artwork CDM-32
CD-ROM disc
 creating CDM-27
CD-ROM image
 preparing CDM-13
 preparing optimized image CDM-16, CDM-17
 required files and folders CDM-12
 testing CDM-20

CD-ROM mastering CDM-1
 application size limitations CDM-6
 error checking CDM-7
 mastering software CDM-4
 media CDM-4
 pathnames CDM-7
 reading integral number of disc blocks CDM-7
 recording speed CDM-24
 software requirements CDM-3
 tips CDM-39
 troubleshooting layout CDM-20
CD-ROM mastering *See Also* Layout tool, Catapult CDM-10
CD-ROM mechanism lifetime CDM-44
CD-ROM recording station CDM-3
CD-ROM space requirements CDM-6
cdrom.tcl file CDM-13
 catapultmegabytes variable CDM-18
 editing for optimized image CDM-18
 editing for simple image CDM-14
 label variable CDM-18
 megabytes variable CDM-18
cdrommaster folder CDM-12
Cluster optimizer CDM-10
Constant linear velocity CDM-40
Creating CD-ROM disc for testing CDM-27

D

data delay CDM-42
data delivery rate CDM-42
data rate issues CDM-41
Disc seeks CDM-41
Double-speed CD-ROM drive CDM-40
Drive specifications CDM-40

E

Editing cdrom.tcl file for optimized image CDM-18
Editing cdrom.tcl file for simple image CDM-14
excludecatapult command CDM-18

F

filemap.out file CDM-19
Files
 reading from CD-ROM CDM-7

G

glob patterns CDM-20

H

Head seeks CDM-44

I

image file *See* CD-ROM image CDM-12

L

label variable CDM-18
launchme CDM-12
layout script CDM-13
Layout tool CDM-10
 cluster optimizer CDM-10
 nonoptimized CDM-10
 troubleshooting CDM-20

M

Mastering software CDM-4
megabytes variable CDM-18

P

Pathnames CDM-7
Preparing optimized CD-ROM image CDM-16,
CDM-17
Preparing simple CD-ROM image CDM-13

Q

QuickTOPiX CDM-4, CDM-25
 setup CDM-24

R

Reading files from CD-ROM CDM-7
Recording speed CDM-24
Relative pathnames CDM-7
Requirements for CD-ROM mastering CDM-1

S

Size limitations for CD-ROM mastering CDM-6
Software requirements for CD-ROM mastering
CDM-3

T

takeme folder CDM-12
Testing performance CDM-43